UNIVERSIDAD POLITÉCNICA DE MADRID
Escuela Universitaria
De
Ingeniería Técnica de Telecomunicación

PROYECTO FIN DE CARRERA

DISEÑO DE UNA APLICACIÓN LABVIEW PARA LA GESTIÓN DE DATOS AUDIOMÉTRICOS

ALVARO M. D. SEIXAS

Noviembre 2002

PROYECTO DE FIN DE CARRERA

TEMA: Instrumentación de Medida.

TÍTULO: Diseño de una aplicación LabVIEW para la gestión de audiometrías.

AUTOR: Álvaro Manuel Domínguez Seixas.

TUTOR: Mariano Ruiz González. **Vº Bº .**

DEPARTAMENTO: Sistemas Electrónicos y de Control (S.E.C.)

Miembros del Tribunal Calificador:

PRESIDENTE:

VOCAL:

VOCAL SECRETARIO:

Fecha de lectura:

Calificación: **El Secretario,**

RESUMEN DEL PROYECTO:

El objetivo principal de este proyecto, es permitir a un amplio conjunto de usuarios, conectarse y mantener interacciones con bases de datos audiométricas desde distantes posiciones espaciales.

Para ello se procederá al diseño una aplicación en LabVIEW, que permita el acceso de usuarios locales y/o remotos a unas determinadas bases de datos audiométricos. Permitiéndoles la ejecución de una serie de funciones, contra las bases de datos, en función de sus privilegios.

Para el desarrollo del diseño se han utilizado las versiones 6.0i y 6.1 de LabVIEW.

Para interactuar con las bases de datos, se ha optado por la utilización del lenguaje de comandos para bases de datos denominado SQL. No obstante, como nuestro programa se diseñara en LabVIEW, ha sido necesaria la inclusión del kit de SQL que posee la propia empresa National Intruments para dicho programa.

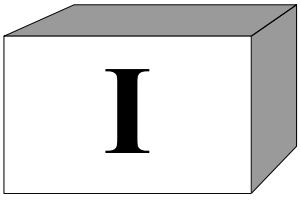
Para la comunicación se emplea el protocolo de transmisión DataSocket que es un protocolo de comunicación punto a punto, que se monta sobre TCP/IP. Este protocolo permite una mayor comodidad y sencillez a la hora de comunicar e interactuar entre dos ordenadores.

Dedicado a mi familia,
novia y amigos

Deseo agradecer a Mariano todo el apoyo, libertad y confianza, otorgados durante el desarrollo de este proyecto.

"Cualesquiera que hayan sido nuestros logros, alguien nos ayudó siempre a alcanzarlos" (Althea Gibson)

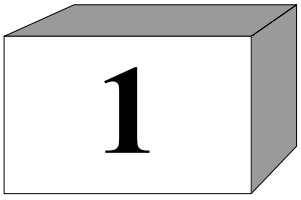
"Aquellos que ven en cada desilusión
un estimulo para mayores conquistas,
esos poseen el recto punto de vista
para la vida."
(Johann Wolfgang Von Goethe)



ÍNDICE

- Índice general

ÍNDICE GENERAL



1 OBJETIVOS DEL PROYECTO

- Definición de objetivos del proyecto
- Fases del proyecto

1. OBJETIVOS DEL PROYECTO

1.1. OBJETIVOS

El objetivo principal de este proyecto, es permitir a un amplio conjunto de usuarios, conectarse y mantener interacciones con bases de datos audiométricas desde distantes posiciones espaciales.

Para ello se procederá al diseño una aplicación en LabVIEW, que permita el acceso de usuarios locales y/o remotos a unas determinadas bases de datos audiométricos. Permitiéndoles la ejecución de una serie de funciones, contra las bases de datos, en función de sus privilegios.

Para el desarrollo del diseño se a utilizado la versión 6.0i de LabVIEW.

Para interactuar con las bases de datos, se ha optado por la utilización del lenguaje de comandos para bases de datos denominado SQL. No obstante, como nuestro programa se diseñara en LabVIEW, ha sido necesaria la inclusión del kit de SQL que posee la propia empresa National Intruments para dicho programa, y que es necesario instalar sobre la versión comercial.

Este kit permite el manejo de bases de datos, a través de comandos en lenguaje SQL, a programas de lenguaje abierto G, como es en nuestro caso LabVIEW.

Como las comunicaciones se harán a distancia, es necesaria también la utilización de un protocolo de transmisión como podría ser TCP/IP.

DataSocket es un protocolo de comunicación punto a punto, que se monta sobre TCP/IP. Este protocolo permite una mayor comodidad y sencillez a la hora de comunicar e interactuar entre dos ordenadores.

Teniendo en cuenta las ventajas anteriormente citadas, DataSocket será el protocolo de comunicación que se utilizará para la comunicación entre los terminales de usuario y el servidor de datos. El hecho de la existencia de más de un terminal, habrá de ser tenido encuentra a la hora de trabajar con DataSocket, que en principio no distingue entre unos y otros terminales.

Por último hacer notar que las bases de datos contendrán información acerca de los pacientes, sus ambientes personales, sus hobbies, sus vicios, etc. A parte de los datos audiométricos.

La aplicación global consta de un servidor que se encargará de dar acceso a las bases de datos, a un número variable de terminales.

Para ello el servidor se encarga de:

- Registrar y asignar privilegios a los usuarios que pidan acceso al sistema. Para la implementación de esta función se diseñará un subprograma denominado Host.vi.

- Crear, actualizar, listar y borrar usuarios. Esta función sólo estará permitida a usuarios que posean privilegios de gestor y para su implementación se diseñará un subprograma denominado Appadmin.Vi. El acceso a este módulo solo sera posible a través del módulo principal que será Host.Vi.

- La creación, actualización, separación, unión y eliminación de bases de datos, que estará condicionada a una serie de privilegios, y que será implementada a través del subprograma DBadmin.VI. Este subprograma, al igual que Appadmin.Vi sólo es accesible a traves de Host.

- Y por último, al gestión de la comunicación con los usuarios que se implementará dentro de un módulo articulado denominado Clentmng.Vi, que se apoyará en otros subprogramas para la gestión de peticiones de pantalla, y las comunicaciones usuarios–servidor y usuarios–bases de datos.

El terminal se encarga de:

- La petición y recepción de datos con el servidor mediante el subprograma Terminal.VI

- El manejo de la interfaz de peticiones así como del de visualización de resultados, a fin de que el programa sea sencillo e incluso intuitivo para el usuario, mediante el subprograma User Interface.VI

1.2. FASES

Las fases de las que consta este proyecto son:

I. Estudio de la herramienta software empleada para la implementación de la aplicación.

II. Estudio de las especificaciones del sistema a desarrollar y trazado de las líneas principales de diseño.

III. Modificación y ampliación de las prestaciones del software.

IV. Realización de pruebas.

V. Redacción de la memoria.

2 INTRODUCCIÓN

- Introducción al sistema realizado.

2. INTRODUCCIÓN

El objetivo de este proyecto fin de carrera es el diseño de una aplicación LabView que se encargará de gestionar el acceso a una base de datos médicos de manera local y/o remota permitiendo la introducción y exportación de los resultados de ensayos audiométricos realizados de una manera muy automatizada.

A modo esquemático, el sistema podría reducirse por una parte a una aplicación de control acceso de usuarios a un entorno dado, y por otra a una aplicación de gestión de bases de datos, las cuales trabajan en conjunto a fin de mantener la seguridad de los datos almacenados, manteniendo su confidencialidad y privacidad.

Al sistema anterior se conectaría una serie de usuarios mediante una interfaz de gestión.

Sin embargo el proyecto no se queda en un entorno teórico de programación en general, sino que está vinculado a una serie de proyectos destinados a mejorar las condiciones de trabajo de los otorrinolaringólogos que han de realizar ensayos audiométricos a sus pacientes.

El sistema global permite a los médicos realizar todo tipo audiometrías de un modo automático. Los resultados de estas audiometrías se obtienen directamente en el ordenador. Una vez realizadas las audiometrías, estos datos se almacenan en una base de datos que se aloja dentro de un servidor con conexión a Internet. De este modo no sólo se obtiene un sistema más cómodo y mejor a la hora de realizar audiometrías, sino que se ponen los datos de estas audiometrías a disposición de cualquier médico que pueda necesitarlos en cualquier parte del mundo. Así, desaparecen las demoras y problemas que implican los traslados de expedientes médicos, pudiendo atender en las mejores condiciones a cualquier paciente en cualquier lugar del planeta.

Este último apartado es el que se desarrolla en este proyecto. Mediante la aplicación generada, se controla el acceso de los usuarios a los expedientes, evitando que personas no autorizadas puedan disponer de los datos, tanto personales, como médicos de los pacientes. Del mismo modo, se garantiza el acceso a los expedientes desde cualquier punto en el que se disponga de conexión con el servidor. Esta conexión ser realizaría través de Internet, red de área loca, WAN, o por cualquier otra vía de conexión (radio enlace, telefonía fija, telefonía móvil, etc...) sobre la cual se implemente un protocolo TCP/IP que permita el empleo de Datasocket.

Como puede observarse, se trata de una aplicación real con una utilidad inherente, que se espera que puedan servir de base para la realización de sistemas comerciales que permitan una mejor atención médica.

Ahora que ya se conoce la intención de este proyecto puede procederse a clarificar esta breve introducción con una aproximación más visual al sistema.

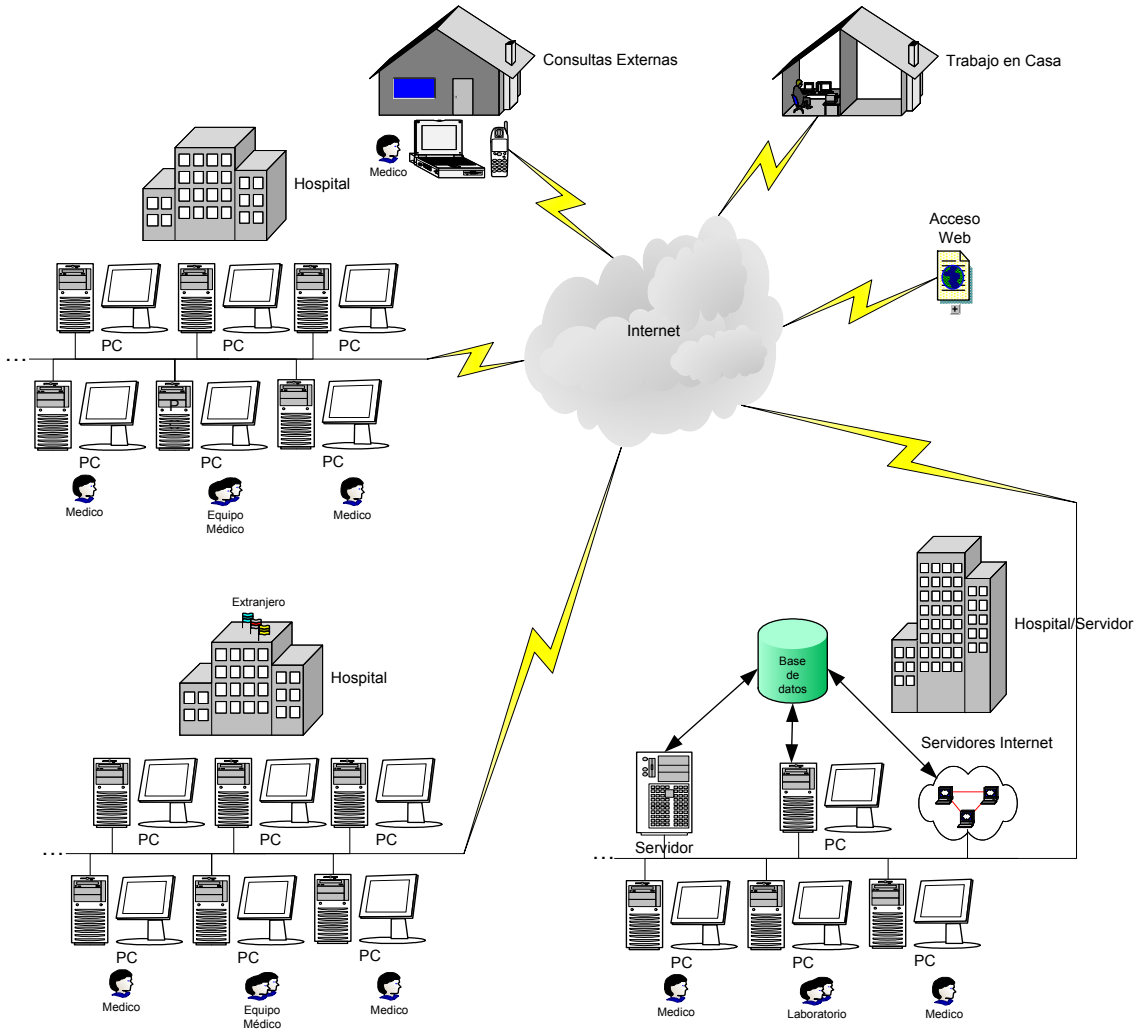
3

VISIÓN GENERAL DEL SISTEMA

- Primera aproximación al sistema.

3. VISIÓN GENERAL DEL SISTEMA

Una primera aproximación a la aplicación seria la siguiente:



En este esquema se observa la versatilidad del sistema, como se mencionó anteriormente la nube de interconexión no tiene porque ser únicamente Internet, sino que, podría utilizarse también cualquier otro protocolo sobre el que se estableciera un sistema IP de identificación de equipos que soportara la utilización de Datasocket.

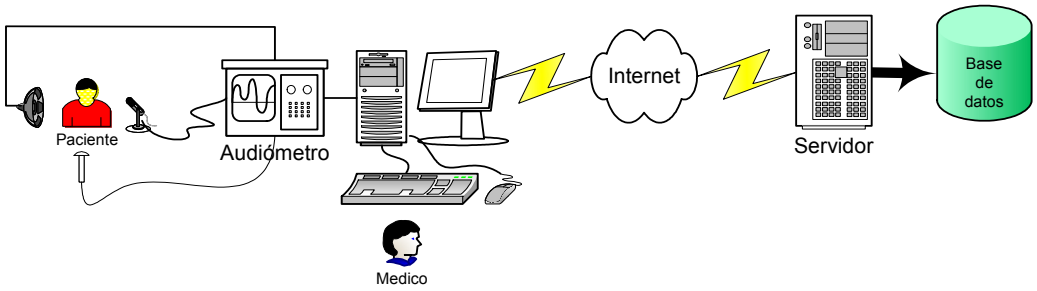
Destaca también la funcionalidad del sistema, que permite el emplazamiento de sedes dentro y fuera del país con un coste de conexión muy reducido, la utilización de una estación propia o compartida con un grupo de usuarios, la conexión a través de red o de línea telefónica, la realización de consultas externas, la realización de trabajo desde casa, la publicación de expedientes en Internet para que los pacientes puedan consultarlos.

Los servidores de Internet servirían para la publicación de expedientes en la red, tema que no se trata en este proyecto.

Por último mencionar que la base de datos puede estar alojada tanto en un servidor, como en un PC normal, a partir de ahora se considerará alojada en un servidor,

opción que da mayor velocidad y seguridad, pero no debe olvidarse que bien podría ser alojada en un PC. Del mismo modo, ese servidor o PC podría emplazarse en cualquier lugar, dentro de un hospital, en una casa, en un equipo de "content hosting" de una empresa externa, etc...

Obsérvese ahora a una presentación menos global y más simple de lo que es nuestra aplicación:



Como puede observarse en la figura, el sistema se puede subdividir en dos partes principales, en la primera está el servidor en el cual se encuentra alojada la base de datos, en ella se gestiona el acceso de los usuarios tanto al sistema como a la base de datos. En la segunda se puede ver uno de los múltiples equipos que conectarían con el servidor. En este caso se trata de un equipo conectado a un audiómetro para realizar los ensayos audiométricos sobre pacientes, parte que no se tratará en este proyecto pero que ha de tenerse presente. Este equipo da acceso a los expedientes almacenados en la base de datos, permitiendo tanto su visualización, como su actualización, en función de los privilegios del usuario utilizado.

Como resultado se obtiene un servidor al que se conectan diferentes terminales de manera simultánea para trabajar sobre la base de datos.

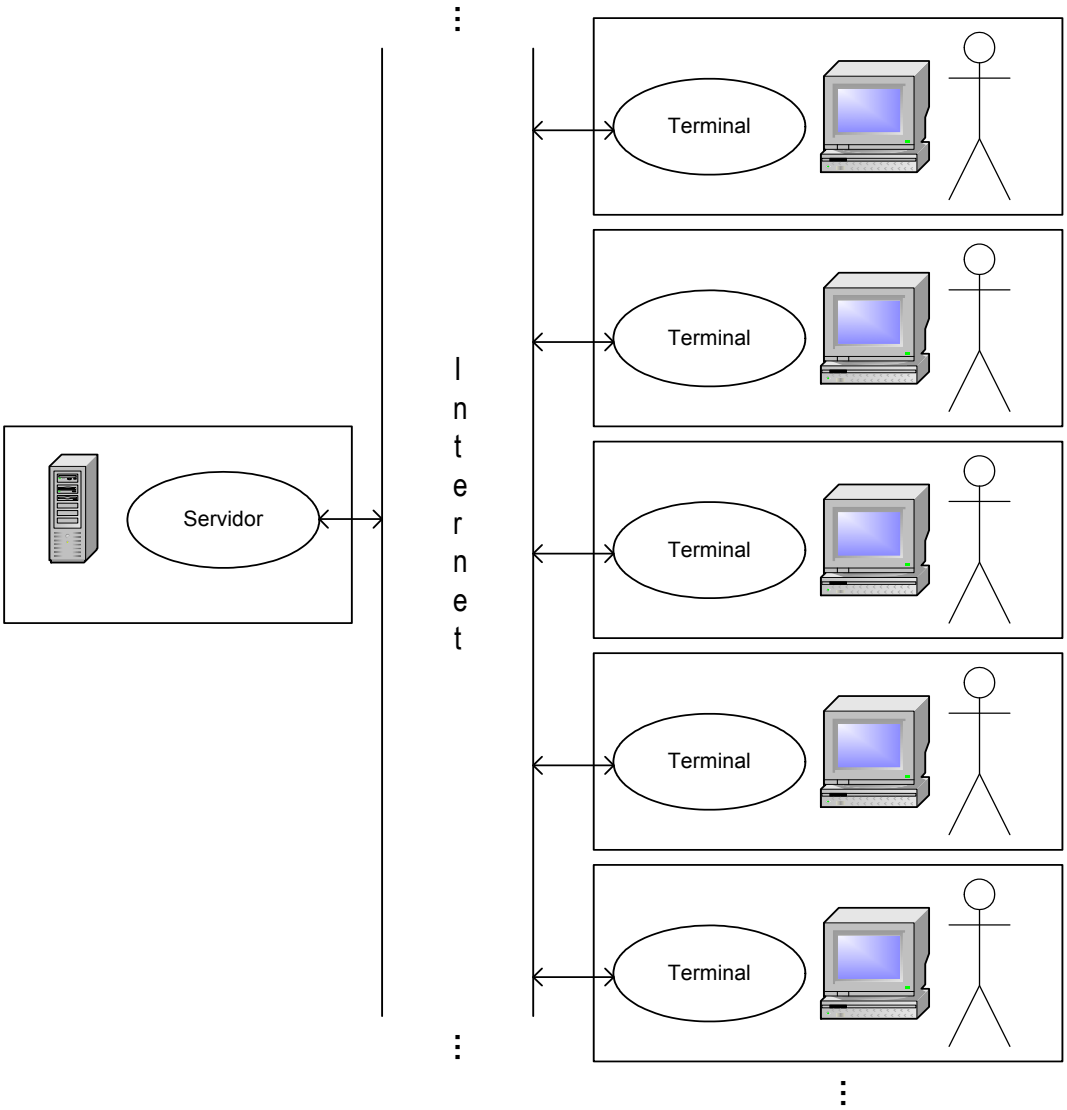
4 EL SISTEMA DE GESTIÓN DE AUDIOMETRÍAS (SGA)

- Descripción de la aplicación.

4. EL SISTEMA DE GESTIÓN DE AUDIOMETRÍAS (SGA)

Ahora que ya se ha visto como es el sistema, es el momento de presentar el SGA, nombre con el que se ha bautizado al software que se ha implementado en este proyecto, y la verdadera razón y resultado del mismo.

El SGA es el software que se encarga de cumplir todas las especificaciones demandadas. Se ha realizado en LabView, y consta de dos partes bien diferenciadas. La primera se denomina "Terminal" puesto que es la que se localiza en los terminales y se encarga de servir de interfaz con el usuario. La segunda se denomina "Servidor" dado que se situará normalmente en un servidor, o en un PC que actúe como tal, y en él se almacenará la base de datos y se encargará de gestionar el acceso de los distintos usuarios a la misma.



Para la interconexión entre terminal y servidor se emplea Datasocket, del mismo modo que para la gestión de la base de datos se emplea SQL.

Llegados a este punto, más de uno se preguntará qué es LabView, Datasocket, SQL... y no es de extrañar, pues en este proyecto no se ha limitado a aglutinar todos los conocimientos adquiridos a lo largo de la carrera y aunarlos en un proceso de creación que confirmará a los desarrolladores como ingenieros técnicos de telecomunicación, sino que se ha aprovechado para, a la vez, añadir un nuevo y continuo proceso el aprendizaje, que, además de la experiencia, ha dado lugar a un mayor conocimiento.

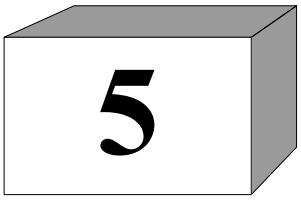
A modo de aclaración puede decirse que LabView es un lenguaje de programación en "open G" desarrollado por National Instruments, empresa americana dedicada principalmente a los sistemas de control. Es un lenguaje de programación visual, es decir, en vez de ir implementando las funciones línea a línea para que luego pueda procesarlo un interprete del lenguaje, todo el proceso de programación se realiza con "cajas" que simbolizan funciones en las que sólo es necesario introducir las variables y direccionar sus salidas para ir procesando la información, obteniendo el resultado deseado, consiguiendo el mismo resultado que en cualquier otro lenguaje de programación pero de una manera más visual e intuitiva, que facilita el trabajo al programador. Por supuesto, para poder realizar el SGA se realizó un curso de auto-aprendizaje con un manual y ejercicios guiados.

Datasocket es un protocolo de comunicación que envía y recibe paquetes de datos entre unas colas de envío, recepción o envío y recepción, que previamente se establecen en un servidor. La decisión de utilizar dicho protocolo, fue poder aprender un protocolo nuevo y su manejo dentro de LabView; si bien, más tarde surgieron ciertos problemas, ya que los equipos quedan pendientes de actualizaciones en las colas, de tal modo que cuando los terminales envían sus peticiones a la cola del servidor no hay problema, pero cuando el servidor envía las respuestas a la cola de terminales, es imposible determinar a qué terminal va dirigida la respuesta. Ante este problema sólo cabían dos soluciones:

1) Crear una cola por cada terminal, lo cual generaba la problemática de establecer una nueva cola cada vez que se activaba un terminal, lo que no era factible pues las colas de Datasocket se predeterminan antes de arrancar el protocolo en el servidor.

2) Añadir un campo de identificación en cada paquete enviado desde un terminal, de modo que cuando los terminales encontraran un nuevo paquete en su cola supieran a quien iba dirigido. Se incidirá en este tema cuando se expliquen las subrutinas de comunicación del SGA.

Por último mencionar que SQL es un conjunto de comandos que pueden emplearse sobre la mayoría de sistemas que gestionan bases de datos, permitiendo manejarlas totalmente, es decir, crear, modificar, eliminar tablas y campos, así como ejecutar todo tipo de búsquedas sobre las mismas. El SQL no es un lenguaje de programación en sí, pero cada lenguaje de programación tiene sus rutinas de ejecución de comandos SQL para poder interactuar con bases de datos sin tener que salir a un entorno de comandos. En el caso de LabView, National Instruments distribuía un paquete denominado NI SQL Toolkit que reunía las mencionadas rutinas de ejecución de comandos SQL, más adelante se actualizó el paquete pasando a ser NI Database Connectivity Toolset, lo que derivó en problemas de compatibilidad que, poco a poco, se han ido subsanando.



EL SERVIDOR

- Arranque.

- Funcionamiento.

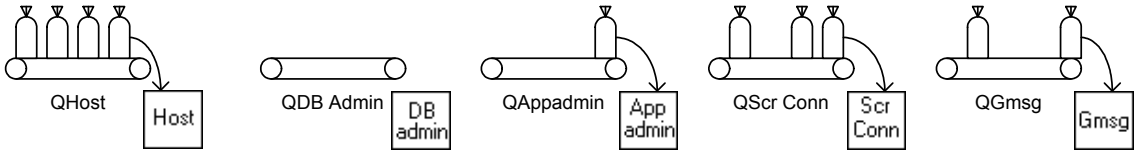
- Rutinas que lo integran.

5. EL SERVIDOR

La parte del SGA alojada en el servidor se encarga de gestionar el acceso de usuarios al sistema, así como del manejo e intercambio de información entre la base de datos y los distintos terminales. Su funcionamiento se describe a continuación:

La rutina de arranque del servidor ("Arrancar Servidor.vi") crea una referencia a la base de datos audiométricos, que almacena dentro de las variables globales, del mismo modo obtiene una cola de datos para cada una de sus subrutinas almacenando también las referencias a dichas colas en las variables globales. Simultáneamente a estas dos acciones, chequea la configuración de datasocket lanzando una aplicación de configuración, en caso de no estar configurado; luego arranca el servidor de datasocket, lo que activa las colas de entrada y salida de datos para la comunicación entre el servidor y los terminales. Mientras el datasocket se activa, se envía una orden de minimización del terminal de estado del mismo, y se visualiza un mensaje de bienvenida.

Una vez inicializado el sistema se activan las subrutinas de gestión de éste que son: Gmsg, Scr_Conn, DS_Pooling, Host, Appadmin y DBAdmin. Cada una de estas subrutinas gestiona una parte especifica del servidor, comunicándose entre ellas por unas colas de datos.



La razón por la que se utilizan las colas es que, al ser el LabView un entorno de programación destinado al manejo de sistemas de control en tiempo real, la programación se orienta a un muestreo continuo, impidiendo dejar una parte del sistema a la espera de la llegada de un dato concreto sin utilizar un bucle infinito de muestreo. Como este sistema sobrecarga el procesador, se utilizan las colas, método que permite dejar una aplicación pendiente de la llegada de un dato a su cola sin tener que efectuar el bucle infinito que carga la CPU.

Como puede observarse DS Pooling no posee cola, esto es debido a que DS Pooling ha de estar a la espera de la llegada de peticiones por la cola datasocket de entrada, y por tanto no puede permitirse que el hecho de tener una cola de mensajes la deje bloqueada a la espera de la llegada de algún mensaje que atender. Es por lo que DS Pooling se dedica a hacer un muestreo de la cola datasocket de entrada, quedando aislada del resto de aplicaciones, a las que sí puede enviar mensajes pero de las que no puede recibir.

Como puede observarse, en principio, cualquier aplicación puede enviar un mensaje a otra, sin embargo para gestionar la mensajería entre colas se utiliza una aplicación concreta por la que han de pasar todos los mensajes. Esto sobrecarga un poco el sistema ya que si una aplicación quiere comunicarse con otra, no puede hacerlo directamente sino que ha de hacerlo a través de GMsg. No obstante este sistema cuenta con la ventaja de poder auditar los mensajes, facilitándose la depuración del sistema. Por tanto, las aplicaciones sólo pueden enviar mensajes a GMsg indicando dentro el mensaje la verdadera aplicación a la que iba dirigida, para que GMsg pueda colocarlo en la cola de mensajes correspondiente.

En este punto es posible hacerse a la idea de que el sistema en funcionamiento consta de varias subrutinas dedicadas a operaciones especificas que interactúan entre sí mediante el envío de mensajes, los cuales pasan todos a través de una aplicación de gestión que los direcciona y audita. Bien, del funcionamiento interno de todas estas rutinas, verdadero corazón del servidor, se hablará más adelante, ahora se terminará de describir el funcionamiento de "Arrancar Servidor.vi".

"Arrancar Servidor.vi" muestra un botón de finalización de la aplicación que al ser pulsado y, tras confirmar el deseo de finalización, manda un paquete de finalización a las subrutinas, destruyendo después tanto la referencia a la base de datos como las colas de entrada de las subrutinas y cerrando el servidor de datasocket.

Una vez descrito el funcionamiento global del servidor (arranque, proceso de peticiones, finalización), se pasa a explicar aspectos concretos de sus rutinas de procesado.

El funcionamiento general del servidor parte de dar servicio a los distintos terminales por lo que puede destacarse en primer lugar las rutinas que se encargan de la comunicación con los terminales. Estas rutinas gestionan las peticiones de los terminales, que llegan a través de las colas de datasocket, hay dos aplicaciones que gestionan dichas colas, una para la cola de entrada y otra para la de salida, en primer lugar se comentará la que se encarga de recibir las peticiones desde los terminales y que es "DS Pooling".

"DS Pooling" es la rutina que se encarga de gestionar la cola de entrada de datasocket. Su funcionamiento es muy sencillo. Esta rutina se queda a la espera de la llegada de alguna petición a la cola datasocket de entrada. Cuando detecta alguno, lo coloca en el gestor de mensajes para que determine a que aplicación iba destinado, y se lo haga llegar. Para que cuando se pare el servidor, esta rutina, que no recibe mensajes de ninguna otra dentro del servidor, se detenga también y no se quede funcionando en segundo plano, se utiliza un mensaje especial de parada, el cual, al ser detectado por "DS Pooling" hace que dicha rutina finalice. Este mensaje especial de finalización, podría ser introducido en la cola por alguna aplicación maliciosa que quisiera impedir que las peticiones de los terminales llegasen al servidor (llegarían pero no serian atendidas), para evitar esto bastaría con introducir un código de seguridad en dicho paquete de finalización. Este código podría ser cualquiera que se estime oportuno, sin embargo, no se considera oportuno codificar el paquete de finalización, bastaría con resetear el servidor para restablecer el funcionamiento normal en caso de piratería.

Siguiendo el procesado de una petición normal, puede observarse que nada más llegar es enviada al gestor de mensajes. Este gestor de mensajes es el denominado "GMsg", el funcionamiento del gestor no es mucho más complicado que el de "DS Pooling". El gestor, al igual que el resto de aplicaciones del servidor, excepto "DS Pooling", tiene una cola de entrada en la que se colocan los mensajes que se envían entre las rutinas internas del servidor así como las peticiones de los terminales recibidas por DS Pooling. Si no hay ningún mensaje en su cola el gestor permanece latente sin ocupar la CPU, y cuando llega uno el gestor analiza el mensaje y determina a que aplicación va destinado. Una vez hecho esto, lo registra en una traza de auditoria y lo coloca en la cola de entrada de la aplicación destino.

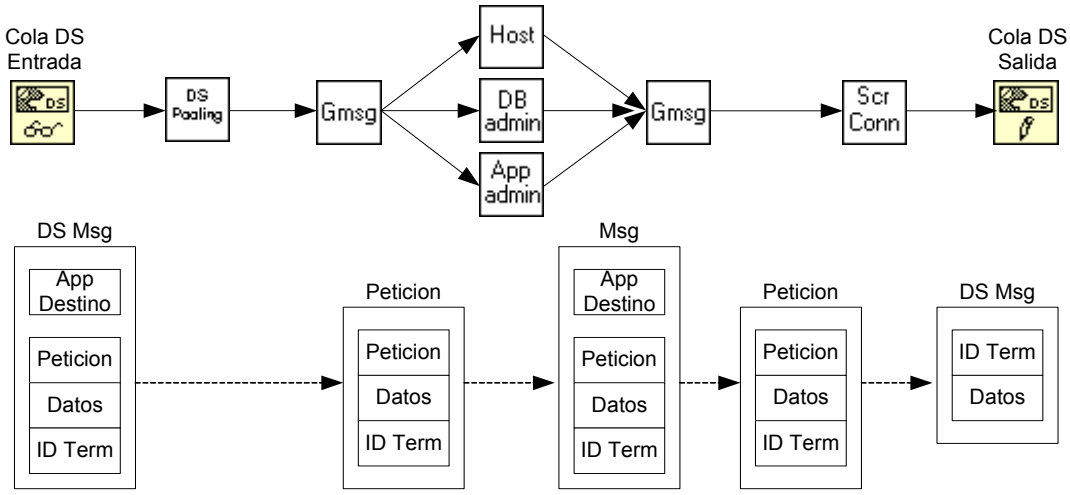
El mensaje enviado por el datasocket contiene una cadena de caracteres en la que van codificados el destino, el comando, los datos y la dirección IP del terminal que lo envió.

Ese mensaje, se envía tal cual se recibe en DS Pooling a GMsg, el cual decodifica una parte extrayendo el código de la aplicación destino, y colocándolo en la cola correspondiente. Nótese que todo lo que se encola, ya sea en datasocket o en las colas de proceso de las aplicaciones, está codificado en una cadena plana de caracteres. Dicha aplicación decodificará el resto del mensaje para obtener la petición, los datos correspondientes a la petición, y la Ip del terminal que realizó la petición.

Una vez decodificado, el mensaje será procesado por la aplicación que generará a su vez un mensaje de respuesta para el terminal. Dicho mensaje es enviado al gestor de mensajes que lo redirecciona a "Scr Conn". Scr Conn es la rutina encargada de la gestión de la cola de salida de Datasocket, enviando las respuestas de las peticiones a cada terminal. Su funcionamiento es inverso al de "DS Pooling" con la salvedad de que al tener cola de mensajes puede ser finalizada de manera interna, sin tener que colocar un mensaje de finalización en el datasocket. Cada vez que un mensaje llega a "Scr Conn" este se vuelca en la cola datasocket de salida del servidor.

Revisando lo mencionado anteriormente puede verse como los mensajes que entran por el datasocket los recibe "DS Pooling" el cual los pasa al "GMsg" que los coloca en la aplicación destino. Ahí son procesados y se genera una respuesta que, de nuevo a través de "GMsg", llega a "Scr Conn" el cual la remite de vuelta al terminal que origino la petición.

En la siguiente figura puede observarse el procesamiento de un mensaje normal, desde su entrada por la cola de entrada hasta su puesta en la cola de salida.



Como se ve el procesado es lineal, el mensaje entra se efectúa y se envía a Scr Conn para su salida. Sin embargo analizando la estructura de los mensajes, una vez procesado por una aplicación, el mensaje puede ser enviado a otra distinta de Scr Conn, y ser así procesado por múltiples aplicaciones hasta que este listo, momento en que se pondría en Scr Conn. Este planteamiento da mucha versatilidad al sistema que aunque en nuestro caso la totalidad de peticiones que acceden a nuestro servidor se resuelven en primera instancia por una única aplicación, permite realizar procesamientos coordinados por las distintas aplicaciones. Aun así, se utiliza el planteamiento lineal, ya que la totalidad de nuestras peticiones se resuelven en primera instancia no precisando mayor proceso y siendo por tanto devueltas directamente a su terminal demandante.

Nótese también que el mensaje de entrada tiene ya la referencia de la aplicación a la que esta dirigida la petición, esto implica que el terminal sabe qué aplicación es la que ha de procesar su petición. Este sistema agiliza el proceso de peticiones, si bien, el terminal ha de ser desarrollado por alguien con conocimientos de la estructura interna del servidor. Otra forma de plantearlo hubiera sido con una tabla de direccionamiento de peticiones dentro del servidor, de modo que el terminal hiciera una petición sin saber que parte del sistema ha de procesarla, y el servidor la direccionaría a la aplicación encargada a partir de la tabla. Esto generaría un sistema más abierto, pero más lento, y como en este proyecto se desarrollan también terminales se toma por la primera opción, dando siempre prioridad a la velocidad del sistema.

Ya se sabe como funciona el sistema de peticiones y la función que desempeñan “DS Pooling”, “GMsg” y “Scr Conn”, es momento ahora de conocer que tipos de peticiones existen y que aplicaciones las realizan.

Bien, si se vuelve atrás al punto donde se enumeraban las aplicaciones que el servidor lanzaba puede observarse que, a parte de las tres descritas anteriormente, están también “Host”, “Appadmin” y “DBadmin”, por tanto, es posible deducir la existencia de tres tipos principales de peticiones, las cuales son procesadas por estos tres elementos.

Antes de describir las peticiones y procesamiento de las mismas, destacar que su proceso se ha repartido en diferentes aplicaciones a fin de mejorar la velocidad, y fiabilidad del sistema, pues si se tratase de una aplicación única que gestionara

peticiones, todas las peticiones se colocarían en una misma rutina que limitaría la velocidad del sistema. Sin embargo, al repartirlas en distintas aplicaciones, el sistema puede procesar varias peticiones simultáneamente, permitiendo a su vez que si, por alguna razón, una de estas aplicaciones cayera, las restantes seguirían funcionando sin problemas. Del mismo modo destacar, que el sistema permite añadir más elementos de proceso de manera sencilla, lo que da pie a la inclusión de nuevas peticiones, así como la adición de nuevos tratamientos a las peticiones existentes.

Volviendo al tema que se trataba, las peticiones se dividen en tres tipos principales, peticiones de acceso al sistema, peticiones de administración de usuarios, peticiones de base de datos.

Las peticiones de acceso al sistema son manejadas por **"Host"** que analiza la petición de acceso realizada con un usuario y una contraseña. La chequea y devuelve el resultado junto con los permisos de dicho usuario. De este modo los permisos de cada usuario son almacenados en cada terminal, esto descarga mucho al servidor, permitiendo que procese más rápidamente las peticiones. Otro modo de operar, sería el de registrar los privilegios de cada terminal dentro del servidor para chequearlos cada vez que un terminal realiza una petición. Esto aumenta la seguridad del sistema, sin embargo como los datos almacenados en la base de datos pueden ser muy extensos, optadose la primera opción que mejora la velocidad a costa de perder un poco de control sobre los privilegios de los terminales. De este modo el control de privilegios se realiza en el lado del terminal y no en el del servidor.

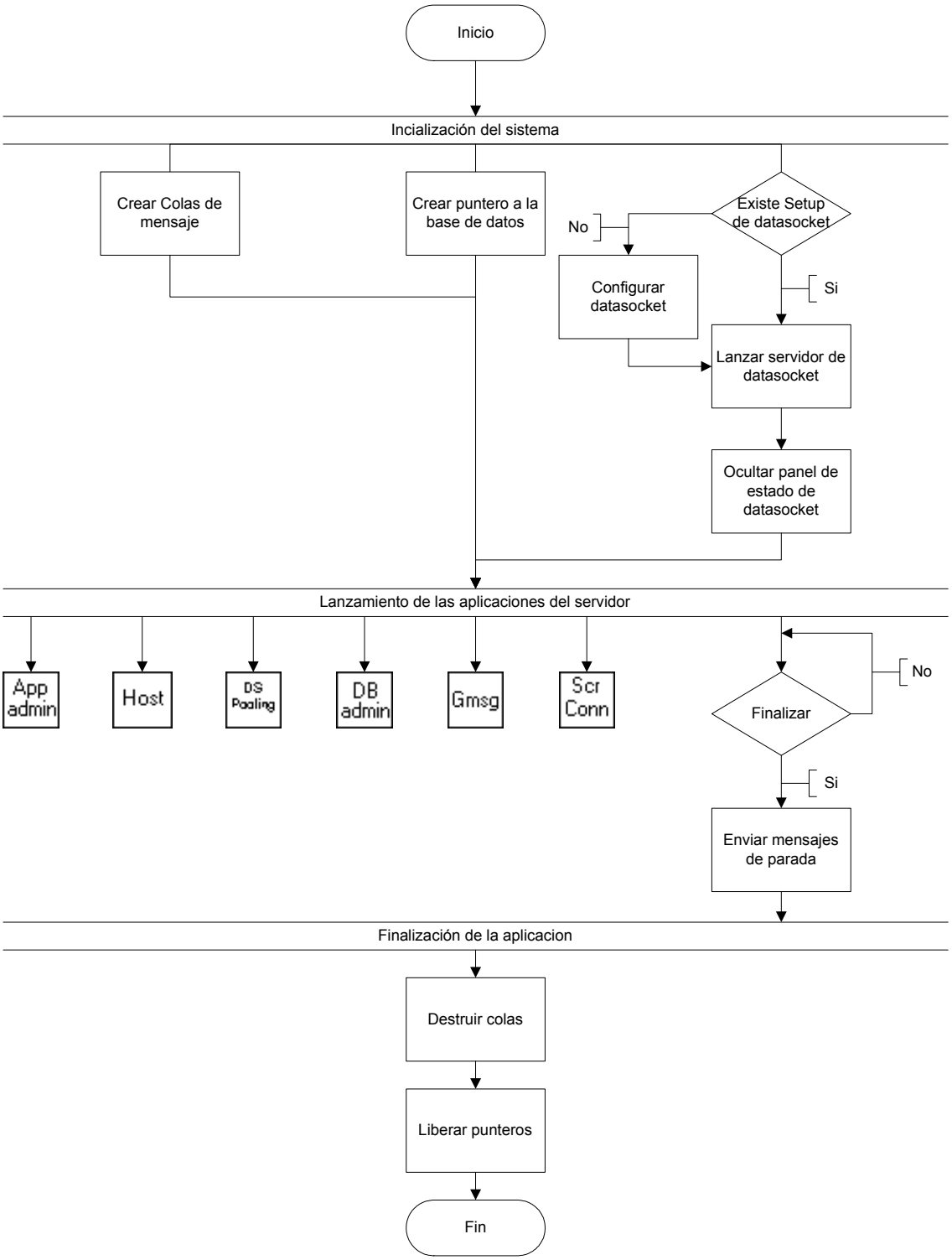
Las peticiones de administración de usuarios son realizadas por "**Appadmin**", esta aplicación soporta las siguientes operaciones: crea, actualiza, lista y borra. La primera crea un usuario con unos privilegios predefinidos para permitirle la utilización del sistema. La segunda, actualiza los datos de un usuario permitiendo aumentar o reducir sus privilegios, así como la modificación de su contraseña. La tercera, devuelve un listado con los nombres de usuarios en el sistema, esta lista permite a los administradores un control más visual sobre los usuarios del sistema. Por último, está la opción que permite borrar un usuario dejándole sin acceso al sistema.

Finalmente se tienen las peticiones de acceso y manejo de la base de datos. Estas peticiones son atendidas y procesadas por "**DBAdmin**". Esta aplicación, buscando una velocidad extrema, implementa un conjunto de peticiones sencillas que permiten el control total de la base de datos. Esto quiere decir que la petición que llega aquí es una petición simple, de modo que, si se desea realizar una función compleja, esta ha de ser construida y realizada por en terminal a partir de peticiones simples, que se van enviando al servidor. Esto diminuye el tiempo de proceso que se dedica a cada terminal, consiguiendo una multitarea más depurada, dando también una, ya mencionada, mayor velocidad.

Ya se conoce el funcionamiento de servidor a un nivel más bien alto, a continuación va a procederse a una descripción más a fondo el funcionamiento de cada aplicación, dando una explicación menos abstracta del funcionamiento, y explicando más detalladamente y con diagramas de flujo, la programación de cada rutina. Se mostrarán siguiendo el orden en el que han sido expuestas en este apartado.

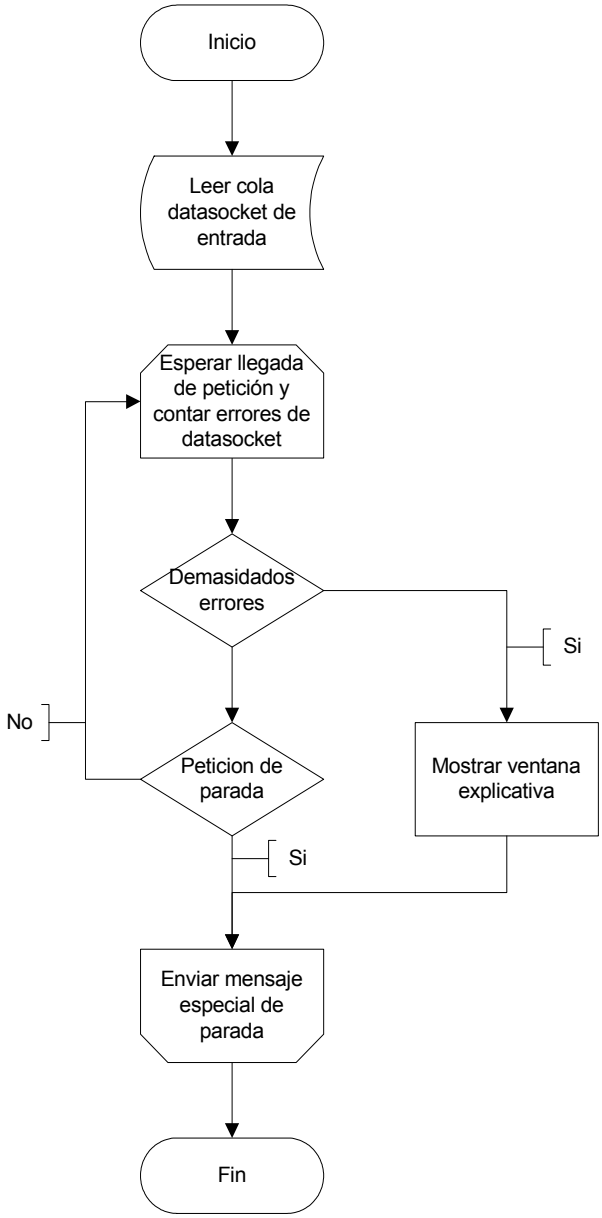
5.1. Arrancar servidor

Como ya se comento este VI es el encargado de inicializar el sistema y lanzar las aplicaciones de gestión del servidor, así como de proceder a la finalización del mismo.



5.2. DS Pooling

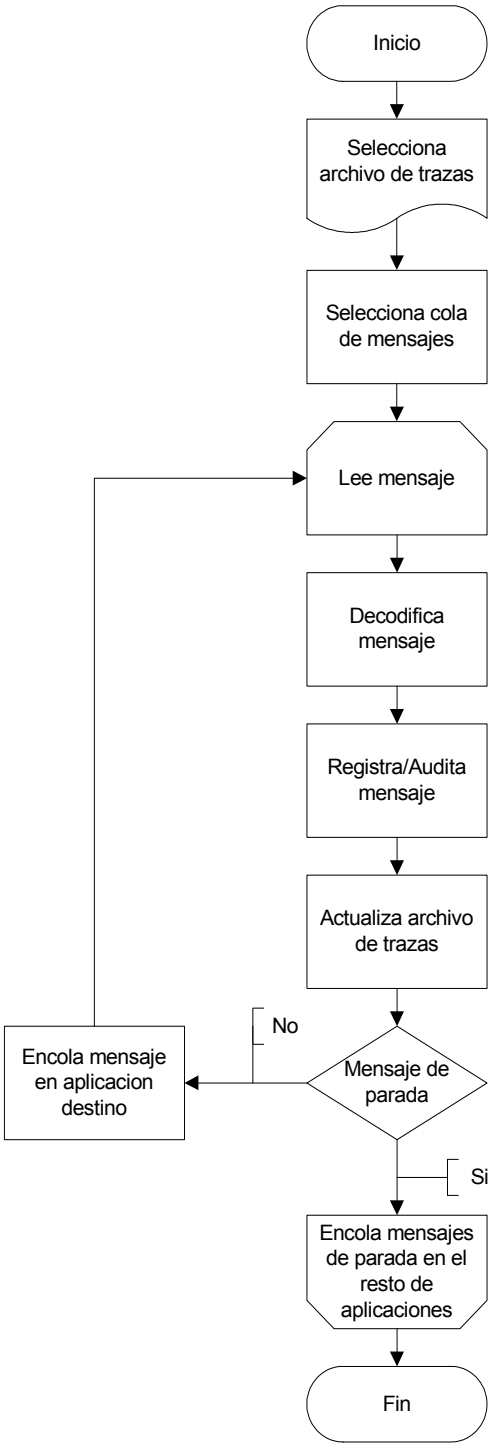
Como se mencionó anteriormente este proceso se encarga de recibir las peticiones que llegan por el datasocket y pasarselos a GMsg para que los haga llegar a su aplicación destino.



El programa lee el nombre de la cola de entrada de datos de la configuración datasocket almacenada en disco y espera la llegada de las peticiones; si le llegan ilegibles u ocurren errores en el datasocket, va incrementando un contador. Cuando el número de errores es demasiado elevado como para poder considerarse normal o bien cuando llega una petición de parada el programa envía el mensaje especial de parada a GMsg y finaliza su ejecución.

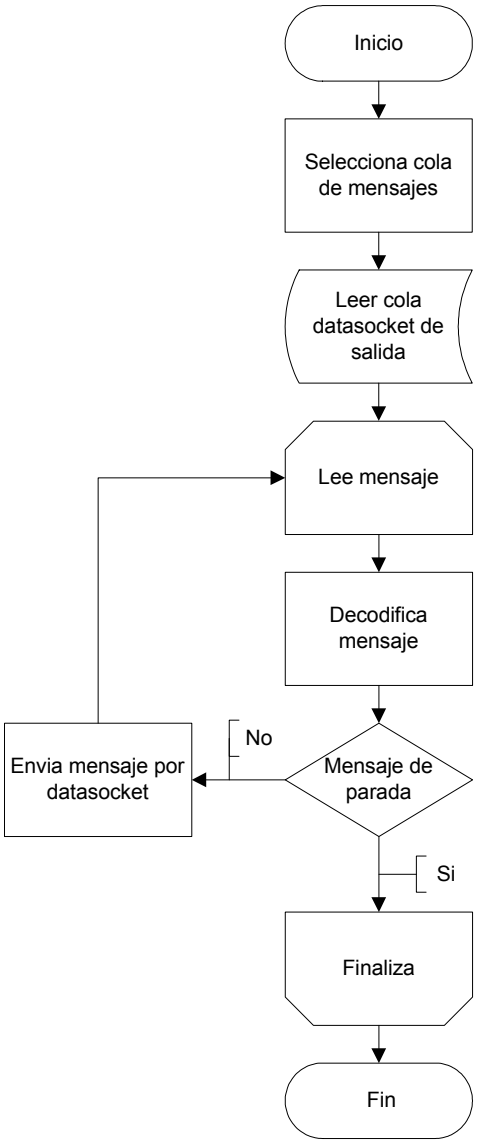
5.3. GMsg

Esta aplicación se encarga de repartir los mensajes entre las distintas aplicaciones del servidor. También audita los mensajes y se encarga de la finalización del resto de aplicaciones.



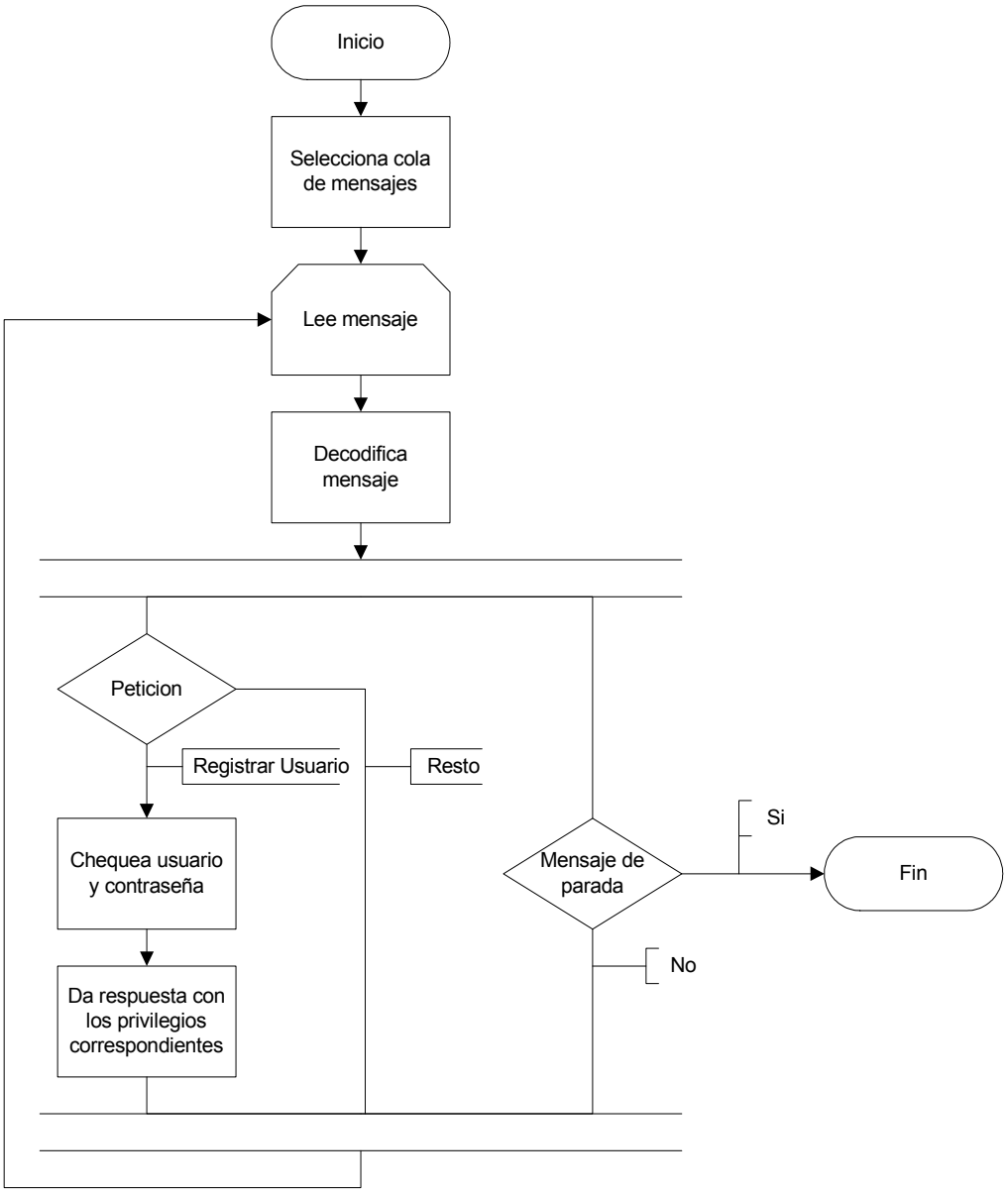
5.5. Scr Conn

Esta aplicación se encarga de enviar las respuestas por la cola datasocket de salida, estos mensajes llevan un código de identificación para que el terminal apropiado lo reconozca como suyo.



5.5. Host

Esta aplicación se encarga de gestionar el acceso de los usuarios al sistema, comprobando sus credenciales y asignando sus privilegios.



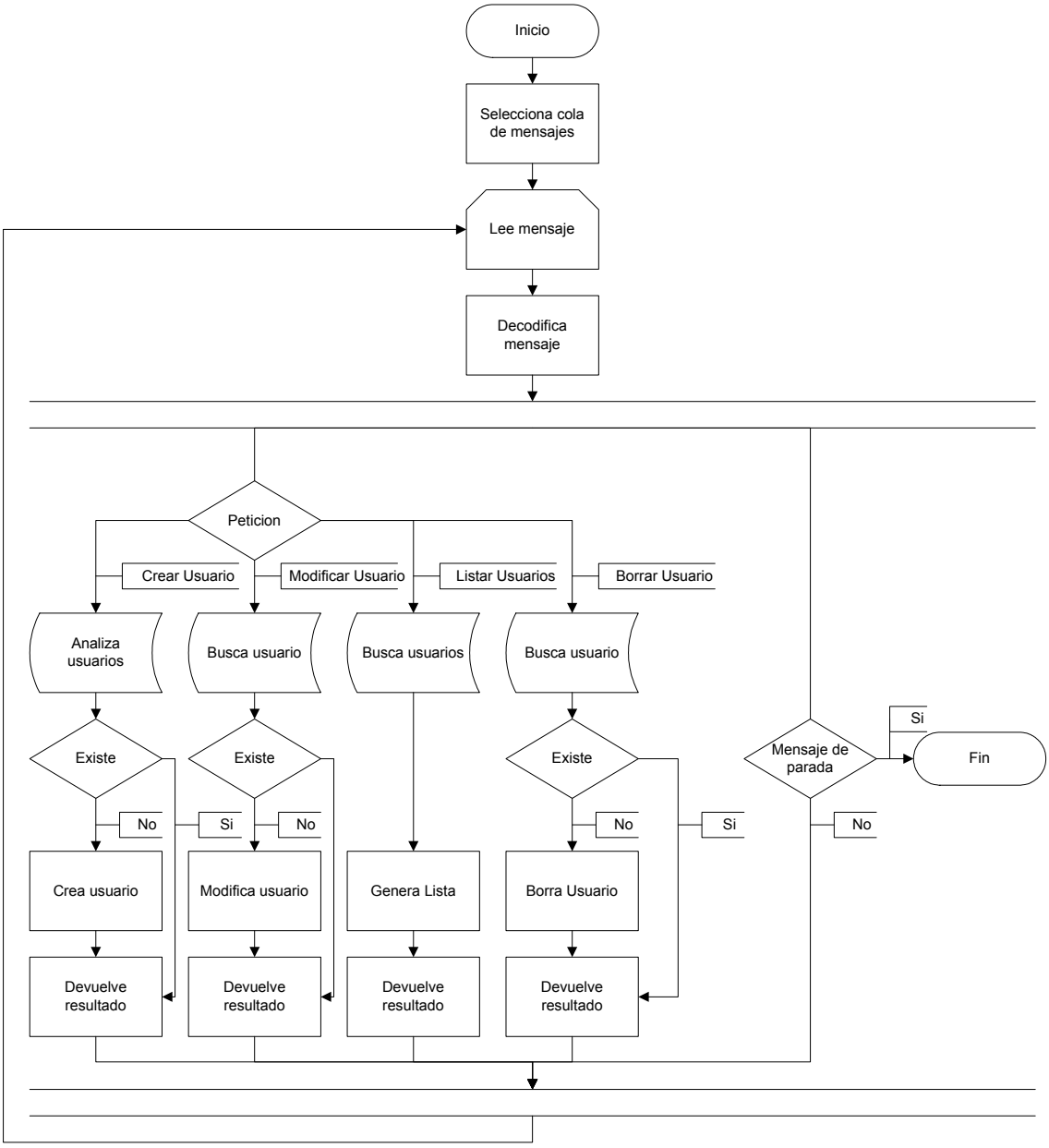
Como puede verse, la única opción que maneja es la de registrar usuarios, las razones de utilizar una aplicación especifica para esta función son :

A) Respetar el sistema planteado en las definiciones iniciales del proyecto, a lo que hay que añadir el hecho de que, si se introdujese dentro de otra aplicación (como por ejemplo Appadmin), las validaciones de los usuarios se realizarían a una velocidad menor.

B) Esta división otorga mayor robustez al sistema. Aun así, podría integrarse fácilmente en cualquier otra aplicación.

5.6. Appadmin

Este programa se encarga de administrar los usuarios del sistema.

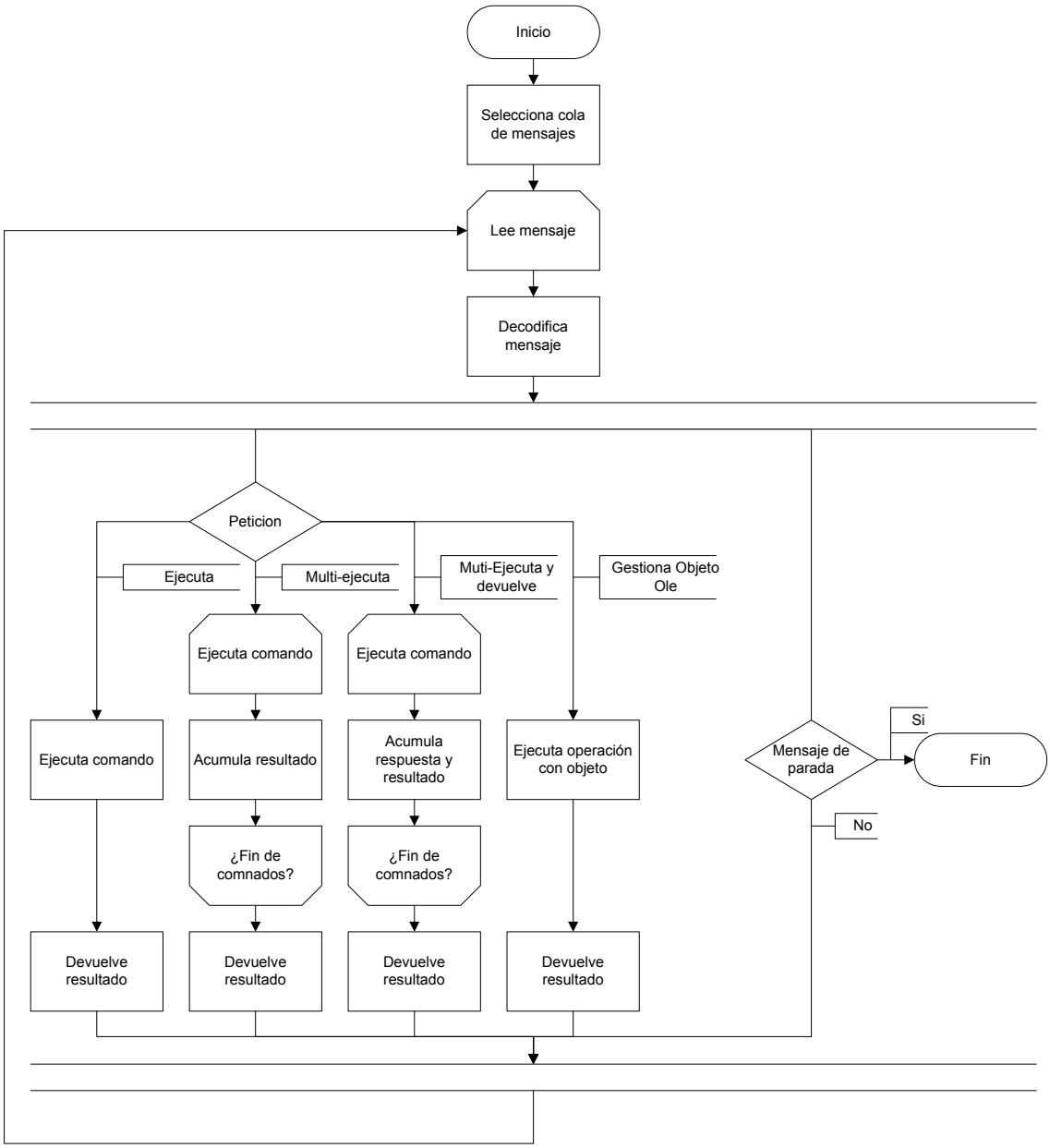


El archivo con los usuarios se guarda codificado en el servidor. Esta aplicación se encarga de la propia codificación del mismo así como de la gestión del archivo.

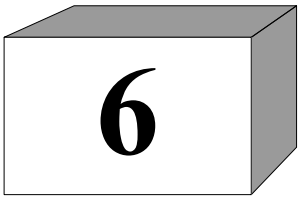
La opción de modificar podría haberse realizado también como combinación de borrar y crear.

5.7. DBadmin

Es la aplicación que se encarga de gestionar la base de datos. Para que esta gestión pueda ser tan amplia como el diseñador del terminal desee, se ha optado por convertirlo en una interfaz de ejecución de todo tipo de comandos SQL simples. Así, conjugando unos comandos con otros pueden realizarse operaciones de cualquier nivel de complejidad, sin sobrecargar el servidor.



La opción que gestiona objeto Ole, se utiliza para trabajar con este tipo de objetos dentro de la base de datos, lo que permite introducir las fotos de los pacientes dentro de la misma, y no mantenerlas fuera de ella, lo que complicaría su gestión al tener que hacer una referencia a archivos externos para localizar las fotos dentro de las búsquedas.



EL TERMINAL

- Aplicación.

- Gestiones que soporta.

- Rutinas que lo integran.

6. EL TERMINAL:

Se denomina terminal a la aplicación generada para servir de interfaz con el usuario, dándole acceso a todos los servicios de servidor, e implementando funciones complejas que satisfagan las demandas del mismo.

El terminal es la parte más compleja del sistema, ya que en el servidor, para garantizar una velocidad máxima, las funciones que se desarrollan tienden a ser bastante simples, siendo por tanto tarea del terminal, manejar dichas funciones para realizar las peticiones complejas que los usuarios demandan. El gasto y carga que esto conlleva en tiempo y procesamiento, tiene una relevancia menor, puesto que el terminal da servicio a un único usuario, mientras que el servidor, tenía que dar soporte a todos los terminales que se conectaran al sistema.

Para su funcionamiento el terminal se apoya en dos aplicaciones principales. La primera actúa de terminal propiamente dicho y la segunda actúa de interfaz de usuario.

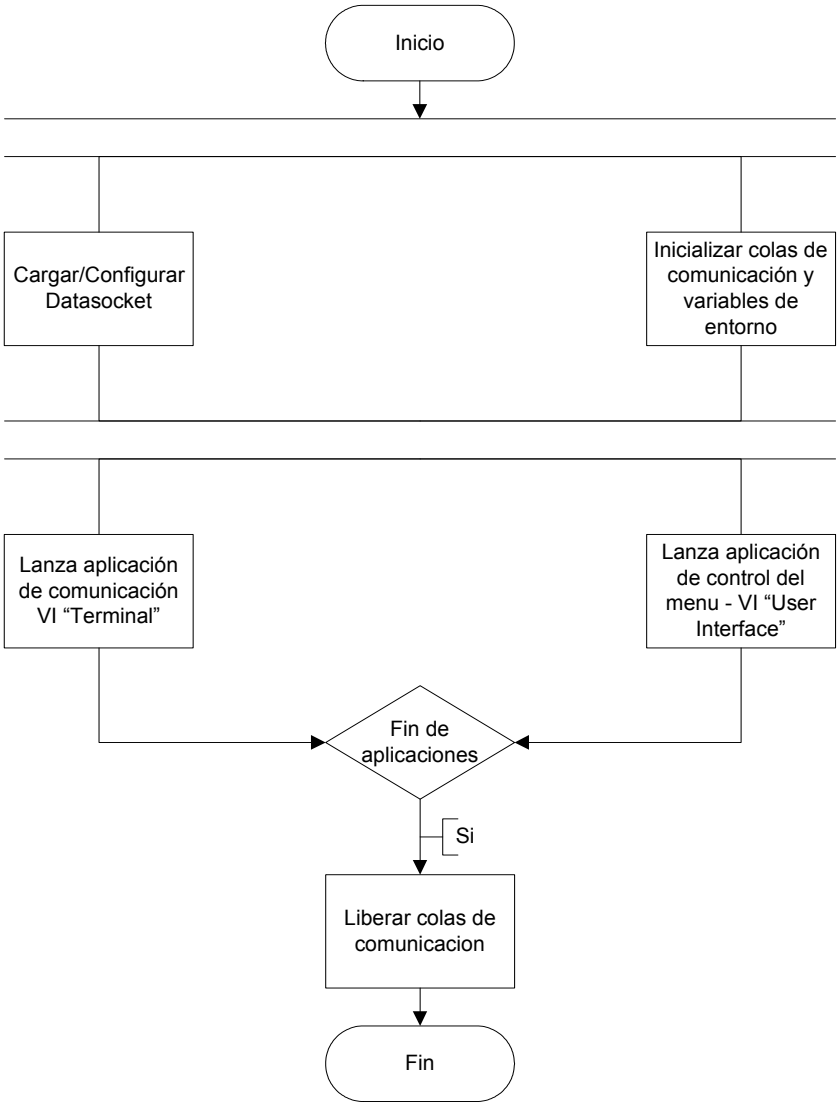
La función de la primera es básicamente la comunicación con el servidor, adaptando los datos, enviando y esperando respuesta. En un principio esta aplicación estaba integrada dentro del interfaz pero tenía un problema, el interfaz se quedaba bloqueado a la espera de la respuesta a su petición, si la llegada de la respuesta a la petición se demoraba mucho, el bloqueo del terminal era muy incomodo, sin embargo al utilizar separar esta aplicación del terminal, se consiguió libertad para continuar utilizando el terminal, y realizar peticiones múltiples, aunque el multiprocesamiento no se ha afrontado a la hora de desarrollar este sistema, de modo que la ventaja radica en que puedes realizar una petición, e ir preparando otra mientras esperas la respuesta a la primera.

El interfaz de usuario se basa en una ventana con un menú desde el que seleccionar las peticiones que se desean ejecutar. Sin embargo este menú delega las operaciones en funciones especificas por lo que su complejidad no es excesiva, hay que tener en cuenta que esta parte del programa es la más extensa y por tanto no se puede afrontar como un todo, sino que se ha ido dividiendo en distintas subrutinas que se encargan de afrontar partes más o menos grandes del entorno a programar.

Para describir esta parte del entorno se comenzará por la aplicación que arranca el terminal y luego se irá ahondando en las distintas subrutinas que vayan apareciendo.

6.1.- VI – Arrancar terminal:

Este VI es el dedicado a la iniciación de la aplicación del terminal, para ello crea las colas de comunicación entre aplicaciones y carga la configuración de datasocket. Aunque la gestión de la interfaz con el usuario lo realiza una de sus subrutinas, el panel que observa el usuario, es el de este VI. En el panel se muestra el menú de opciones que gestiona toda la aplicación.



Aunque pueda no parecerlo la comprobación de fin de aplicaciones lanzadas en LabView se realiza de forma automática, es decir como VI lanza un par de subrutinas, mientras estás estén funcionando el programa no podrá finalizar. Por tanto no existe bucle de comprobación de finalización de subrutinas, de ahí que en el diagrama no aparezca una realimentación con la opción "no", cuando las subrutinas terminen, la aplicación principal retomará el control y finalizará. Todo esto sin haber sufrido una carga superior en CPU.

6.2.- VI – Terminal:

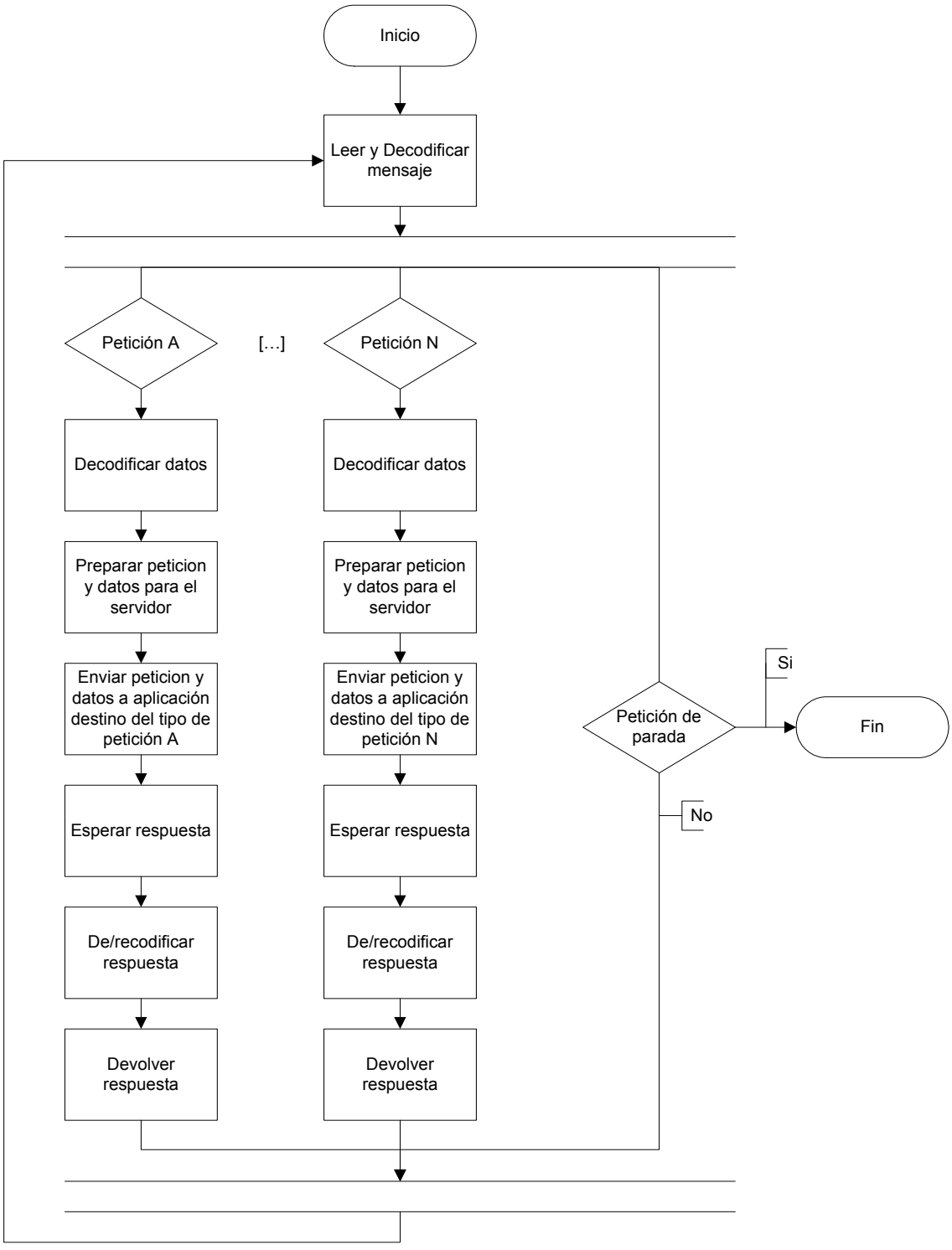
Probablemente el nombre escogido no sea el más adecuado, pudiendo llevar a la confusión, pero esta aplicación se encarga de enviar y recibir las peticiones que realiza el usuario a través de la interfaz. Esta aplicación ha sido diseñada conociendo el funcionamiento del servidor, por lo tanto se sabe lo que se ha de enviar, y lo que se debe recibir como respuesta. Presenta una opción de envió por cada una de las distintas peticiones que pueden enviarse al servidor, del mismo modo, sabe qué aplicación del servidor se encarga de soportar dicha función, y en qué formato han de enviársele los datos, así como el tipo de respuesta y los datos que generará el servidor. De este modo el terminal no realiza ninguna función compleja, pero habilita a sucesivos programadores a diseñar nuevas aplicaciones que trabajen con el servidor sin necesidad de ningún conocimiento sobre el mismo. Simplemente basta con conocer y adicionar el terminal a su diseño. Así las aplicaciones complejas se desarrollan en la interfaz, de la cual se ha creado un prototipo, pero también se deja la puerta abierta para versiones posteriores e independientes del nuestro. Su funcionamiento se esquematiza en el diagrama de flujo representado en la página siguiente.

Como se puede observar el VI Terminal se encarga de recibir peticiones, adaptarlas al servidor, enviarlas, esperar la respuesta, y transformarlas para devolverlas a la interfaz demandante. Este formato de funcionamiento limita la multitarea del sistema, ya que si se quiere realizar más de una petición, estas se colocan en el VI Terminal y se realizan de forma secuencial y no en paralelo. Otra limitación del formato es que sólo permite la conexión de una única interfaz al VI Terminal, lo cual puede ser poco interesante en redes de área local, o bien, en estaciones que lancen más de una instancia de la interfaz.

Para paralelizarlas bastaría con añadir códigos identificativos de número de petición, y para permitir más de una interfaz añadir otro código identificativo más, ya sea por puntero de cola (en el caso de ser varias interfaces en una misma estación de trabajo) o por direcciones IP (en el caso de ser varias interfaces que vuelcan en el terminal a través de una red LAN), de tal modo que el VI Terminal pudiera enviar más de una petición, y luego diferenciar la respuesta en función de los códigos identificativos (interfaz/nº de petición) de la respuesta. Para ello bastaría una pequeña y simple modificación en el servidor, que permitiera adicionar este segundo nivel de código identificativo dentro de las peticiones enviadas.

De todas formas, este complejo sistema de peticiones no se llegó a desarrollar, ya que se asumió que tanto terminal como interfaz estarían en la misma estación, y no se realizaría más de una instanciación del terminal dentro de la misma. Del mismo modo se consideró innecesario añadir un segundo nivel de identificación en las peticiones, no por no ser interesante, sino porque complicaba sobremanera un sistema que ya en principio no se mostraba sencillo, para lograr una multitarea que dentro de esta aplicación no era especialmente útil, ni importante, ni mucho menos critica.

Nuestra interfaz permite el lanzamiento de peticiones en paralelo, pero esto no debe llevar al engaño, pues al pasar estas por la aplicación terminal, pasan a procesarse todas en serie.

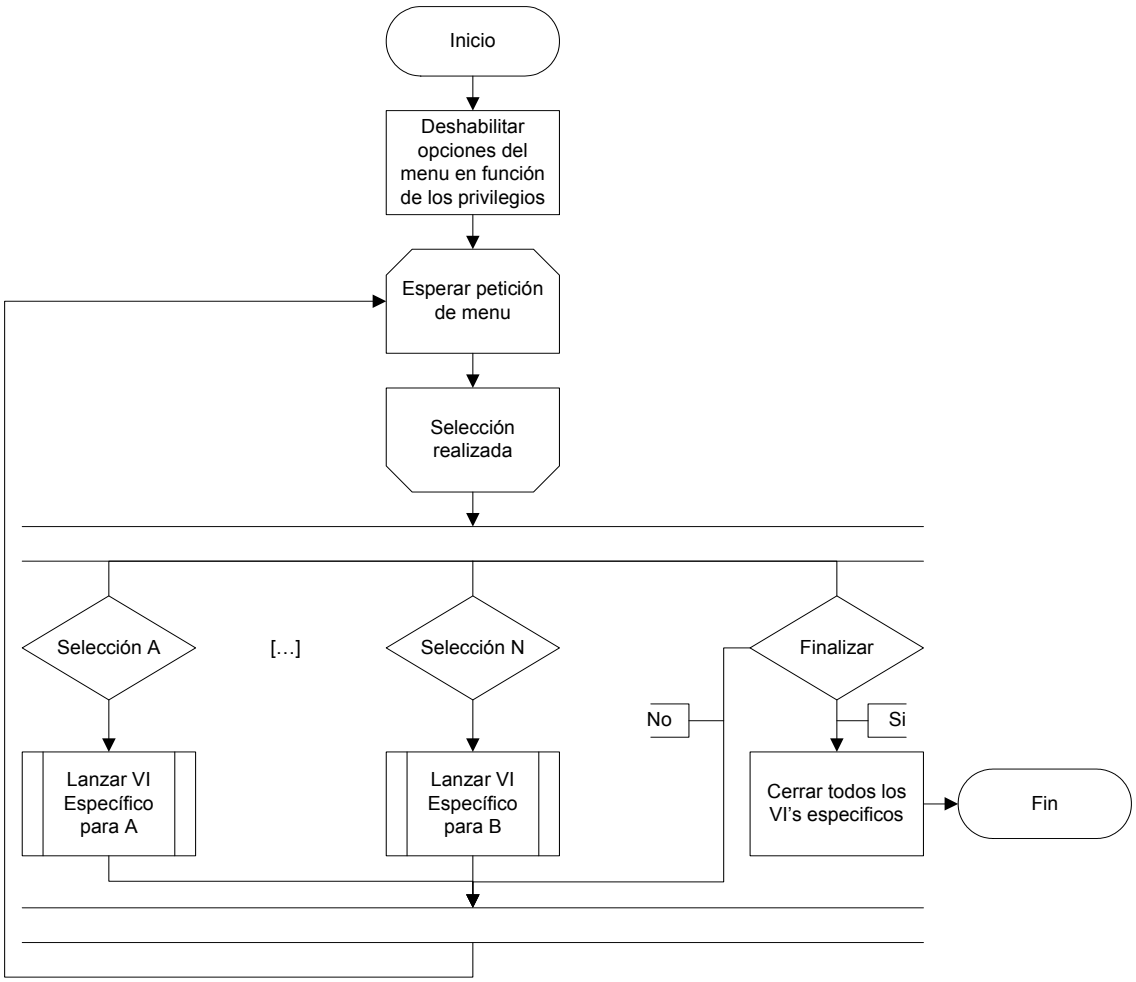


"Puede añadirse una lista de peticiones y clasificación que realiza el terminal"

6.3.- VI – User Interface:

Este VI se encarga de la gestión de la interfaz con el usuario. Recuérdese la interfaz principal no se encuentra dentro de este VI, sino que se aloja dentro del menú, del panel del VI Arrancar Terminal, de ahí el uso del término gestión. Al lanzar el VI Arrancar Terminal, éste muestra un menú con las opciones principales del sistema, sin embargo, el control y la gestión de ese menú la delega en este VI manteniendo la modulación del sistema.

El VI User Interface recibe, al arrancar, una referencia al menú del VI Arrancar Terminal, y es con esa referencia con la que realizará toda la gestión y control sobre las peticiones realizadas a través del mismo.



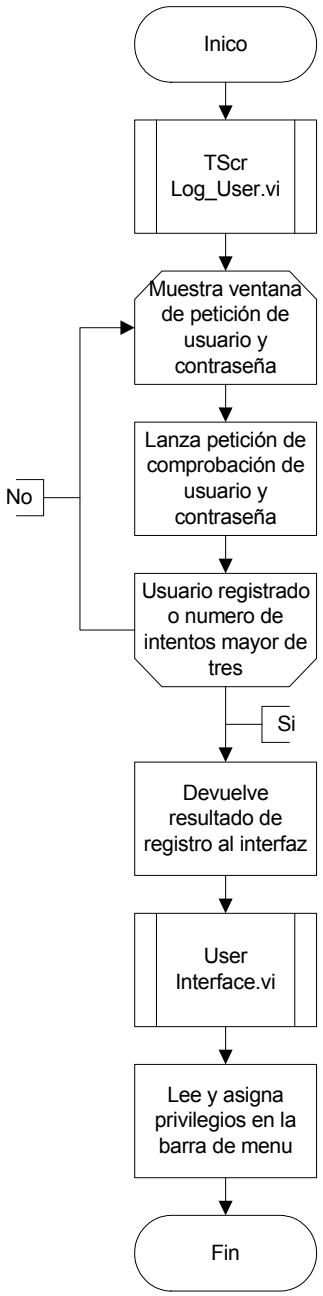
Cada selección tiene su panel de relleno de parámetros para la petición, que se lanza automáticamente, el programa permite abrir varios paneles a la vez, pero recuérdese que, aun así, las peticiones se realizarán de forma secuencial

La aplicación desarrollada dentro de cada uno de los paneles es la encargada de generar la función, más o menos compleja, que se encargue de gestionar la petición demandada de un modo completo y a partir de las funciones sencillas que pone a nuestro servicio el servidor, y que están disponibles a través del VI Terminal. Lógico es pues, el analizar cada uno de estos subprogramas.

6.3.1.- Login

"Login" es la acción de entrada al sistema, en ella se registra al usuario, comprobando su nombre y contraseña, y se le otorga el nivel de acceso que le conceden sus privilegios.

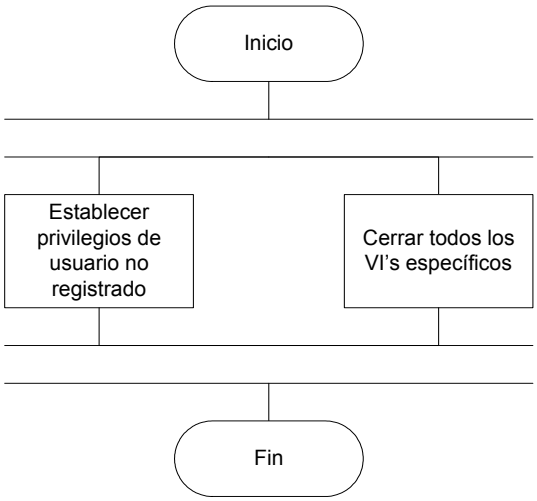
Esta es una de las pocas opciones que no se realiza de manera independiente en una aplicación distinta, sino que depende de la interfaz del usuario. La razón es simple, la aplicación que se encarga de gestionar el menú de la interfaz, es el propio VI User Interface, por tanto para dar acceso, en función de los privilegios, no es posible desvincularse de este programa principal, ya que, para garantizar que se cumpla la escala de privilegios, se analizan los permisos, y en función de ellos se habilitan o deshabilitan las funciones correspondientes dentro del menú, restringiéndolas a usuarios autorizados. El login se desarrolla por tanto de manera conjunta con el VI TScr Log_User, del siguiente modo:



6.3.2.- Logout

La acción de "logout", es la opuesta a "login", en ella el usuario abandona el sistema sin cerrar la aplicación, por lo que pierde todos sus privilegios y queda con las opciones básicas (login, exit, ayuda), esta opción pertenece también a este grupo de opciones especiales, puesto que no se realiza en un VI independiente, sino que se realiza por completo en la interfaz de usuario, esto es debido a que el control de privilegios no lo lleva el servidor a fin de ganar velocidad al no tener que mantener un registro de usuarios conectados, como ya se comentó en su momento. Esto quiere decir que no es necesario avisar al servidor del hecho de que se va a abandonar el sistema, y tampoco es necesario solicitarle nuevos permisos, sino que basta con asignar a la interfaz de usuario, los permisos de un usuario no registrado, y esta queda lista para dar entrada a un nuevo usuario.

Por la misma razón mencionada, una vez registrado un usuario, se puede realizar un nuevo "login" sin necesidad de "deslogarse", ya que el servidor no sabe si el usuario está registrado; al mandarle una nueva petición de registro él devuelve los privilegios correspondientes al usuario solicitado y en caso de no conseguir registrarse correctamente, devuelve los privilegios de usuario no registrado.



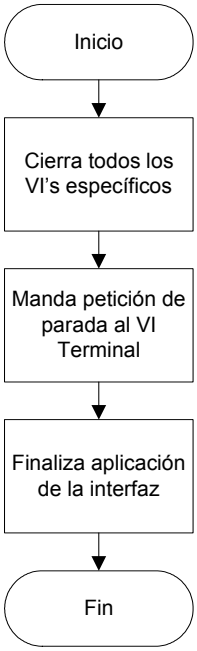
Como puede observarse se cierran todos los VI's específicos, ya que el sistema permite ir abriéndolos en función de los deseos del usuario, y al abandonar el sistema ha de garantizarse que éstos se cierran, impidiendo que alguien, incluso sin necesidad de registrarse, pueda utilizarlos.

Esta operación no fue descrita dentro de "login" pero, aunque no aparezca en el diagrama de flujo, es también realizada, puesto que es posible, como ya se mencionó, volver a registrarse sin necesidad de hacer primero un "logout". Si esto ocurre y no se cierran las aplicaciones, el nuevo usuario podría tener acceso a ventanas que el anterior hubiera dejado abiertas, y para las que en principio podría no tener permiso, pero que mientras que estuvieran abiertas podrían ser utilizadas por este nuevo usuario.

6.3.3.- Exit

Esta acción cierra la aplicación del Terminal de usuario y al igual que las dos anteriores, no se realiza en un VI independiente.

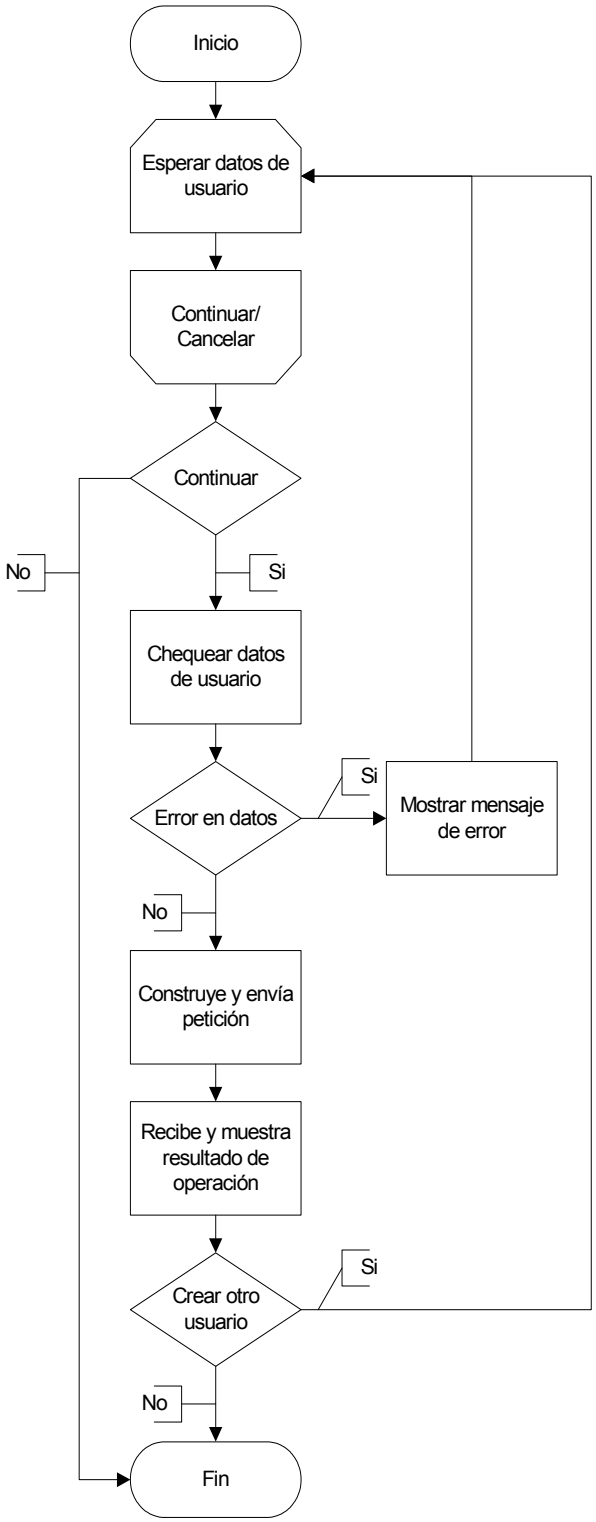
En ella, se manda el paquete de parada al VI Terminal, se cierran todas los VI's de gestión específicos, y se cierra la interfaz de usuario, por lo que el VI Arrancar Terminal retoma el control y cierra la aplicación global.



6.3.4.- Crear usuario

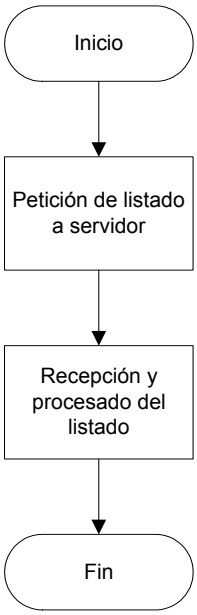
Esta acción crea un nuevo usuario, mediante la asignación de un nombre, una contraseña y una serie de permisos específicos para este usuario. Toda la operación se lleva a cabo por la subrutina VI TScr C_User.

El procedimiento es el siguiente:



6.3.5.- Listar usuarios

Esta operación muestra una lista con los usuarios y privilegios existentes en el sistema. La operación la lleva a cabo el VI TScr L_User mediante el siguiente procedimiento:

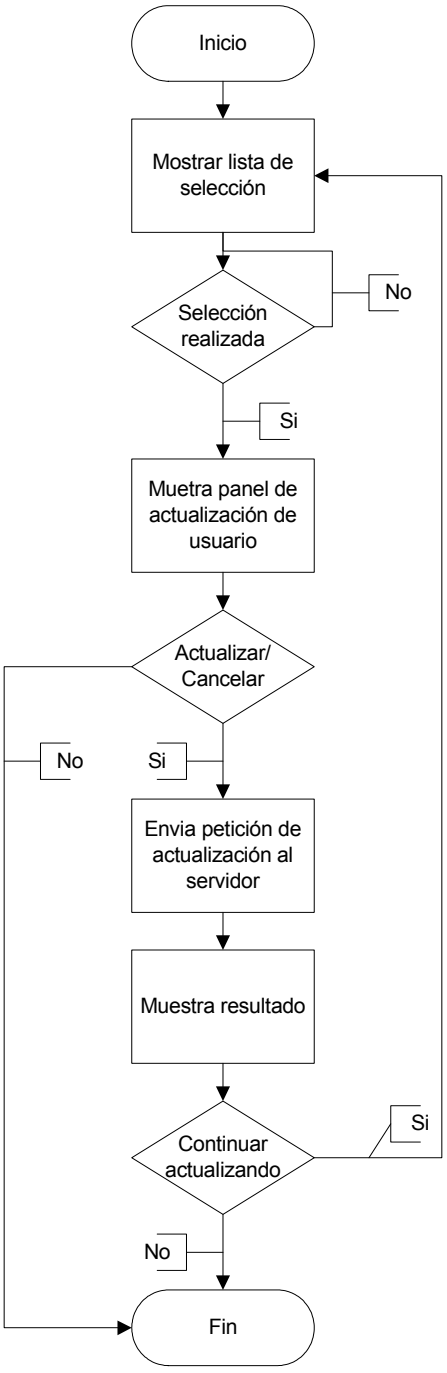


Como puede observarse, la aplicación solicita un listado de usuarios al servidor, el cual responde enviándolo, sin embargo en el servidor los usuarios se almacenan por orden descendente de creación, por tanto, el listado recibido se ordena por nombre de usuario, y se muestra en una tabla de selección.

Este proceso tan sencillo, se emplea en el próximo VI para seleccionar el usuario a modificar.

6.3.6.- Modificar usuario

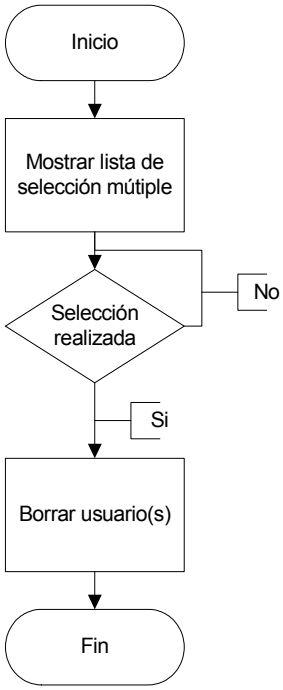
Esta opción permite modificar los datos de un usuario, en principio permite modificarlos completamente, tanto nombre de usuario, como contraseña y/o permisos. Se realiza en el VI TScr M_User y su funcionamiento es el siguiente:



Nótese que si bien es posible modificar completamente el usuario, no se puede introducir el usuario a modificar manualmente, sino que es necesario usar una lista de selección, esto se debe a que un usuario puede tener el mismo nombre, pero distintas permisos para distintas contraseñas, lo que hace que el mejor modo de determinar el usuario a actualizar sea mediante la selección en un listado, ya que manualmente los datos a introducir son demasiados y complicados de obtener, siendo fácil equivocarse.

6.3.7.- Borrar usuario

Esta opción permite la eliminación de uno o varios de los usuarios del sistema. La aplicación que la gestiona es el VI TScr D_User.



Como puede observarse el programa muestra una lista de selección similar a la mostrada en actualizar usuario, si bien, en este caso puede realizarse una selección múltiple a fin de eliminar más de un usuario de una sola vez.

La razón del uso de la lista de selección es la misma descrita en el apartado anterior, es la manera mejor y más fácil de determinar el usuario y privilegios concretos que se quiere eliminar.

También comentar, que el listado se obtiene vía petición al servidor de una manera casi idéntica a la realizada en "listar usuarios". De este funcionamiento puede deducirse que cada uno de estos VI's independientes se comunica con el servidor de manera transparente para la interfaz de usuario, ya que envían sus peticiones directamente al servidor a través del VI Terminal, sin devolver el control a la interfaz, que por su parte estará esperando ordenes por parte del usuario, por si desea abrir alguna otra ventana adicional a la que esta utilizando, o cualquier otro tipo de petición (login, logout, etc...).

6.3.8.- Crear base de datos

El VI encargado de crear bases de datos es el VI TScr C_DB; este VI genera una base de datos con el nombre deseado por el usuario, sin embargo hay que tener en cuenta el método de creación.

SQL es un lenguaje de comandos con el que, mediante un intérprete, puede ejecutarse sobre una base de datos dando lugar a operaciones y resultados. El lenguaje no estaría completo si no fuese capaz de crear tablas, comandos, etc. Sin embargo, esta capacidad no es lo que verdaderamente le hace potente, pues hoy día se cuenta con gran cantidad de programas distintos para la gestión de base de datos que permiten su creación de una manera mucho más sencilla que vía SQL, el problema es que esas bases de datos no suelen ser compatibles con las de otros programas, pero si mantienen siempre compatibilidad con SQL.

De este modo, se puede crear una base de datos con cualquiera de estos programas, de tal forma que luego se pueda emplear SQL para trabajar con ella, sin haber tenido que crearla con el procedimiento complejo que para ello se requiere mediante SQL.

En este caso se escoge la herramienta ***Access de Microsoft***, que permite crear la base de datos de una manera, más visual y sencilla a la necesaria por SQL, con la ventaja de que al ser creada como base de datos de Access puede consultarse su contenido de manera directa con dicho programa, facilitando el depurado del programa.

Como el proceso de creado de la base de datos en Access a de ser realizado directamente por el usuario, podría volverse a pensar que para la creación de una nueva base de datos sería necesario que el programa lo realizase vía SQL o entonces que el usuario tuviera suficientes conocimientos, tanto de la base de datos, como del programa, para crearla en Access u otro programa de gestión de bases de datos. Sin embargo como ya se ha mencionado que el proceso de generación SQL es demasiado largo, se ha optado por obtener una copia de la base de datos generada en Access y guardarla como plantilla vacía.

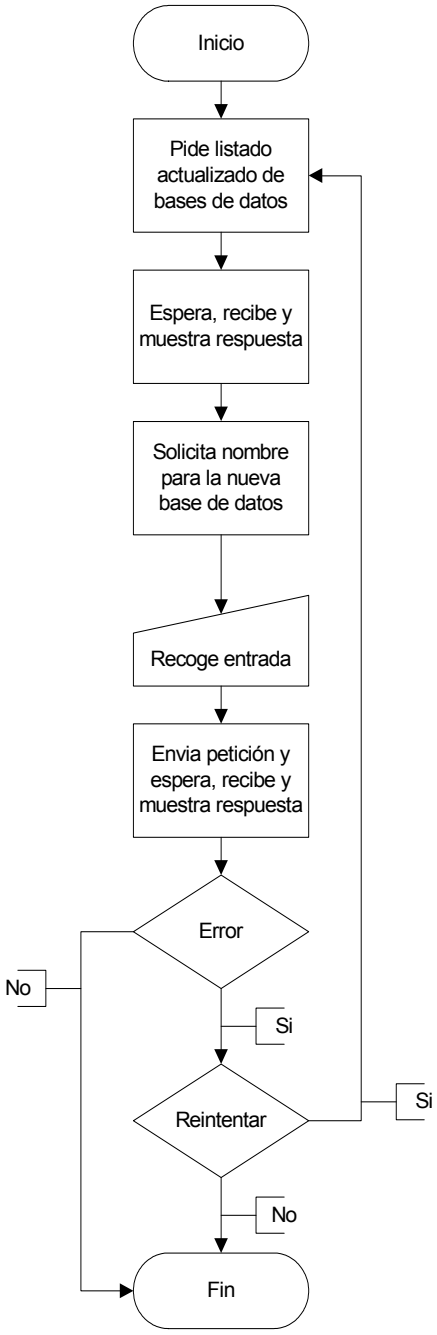
Así, cuando se desea crear una nueva base de datos, basta con hacer una copia de la plantilla. Esto impide añadir campos o modificar las tablas, ya que todas las copias de la plantilla son idénticas, pues el uso de base de datos se basa en estructuras fijas que no van a cambiar con el tiempo, siendo esta característica la que las hace tan fiables, y permite ejecuciones de gestión sobre unas cantidades grandes de datos. De hecho el programa esta orientado a esta base de datos concreta, pues se descubrió una serie de rutinas que permitían trabajar con las bases de datos como algo dinámico, pudiendo modificar su estructura y tipo de datos en cualquier momento, pero al intentar utilizarla, resultó que en realidad se perdía mucha velocidad y aún más fiabilidad en el sistema.

El tema de la base de datos se trata más a fondo en el capítulo 6.

Los comandos SQL para creación de una base de datos con sus tablas y campos concretos, pueden encontrarse en el anexo correspondiente a SQL.

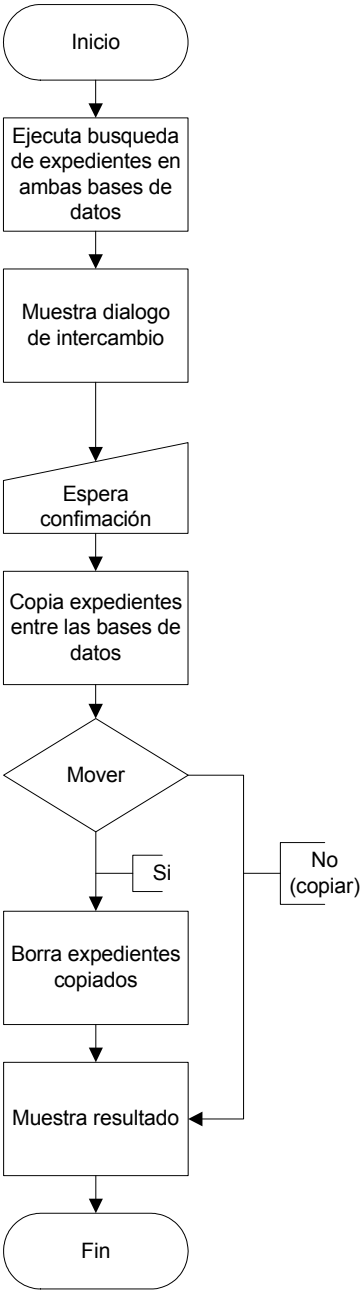
Como se ha observado, se ha utilizado una plantilla para la generación de nuevas bases de datos, por lo que ahora resta saber cómo se codifica la rutina que implementa la transformación de la plantilla en una base de datos activa.

La operación es sencilla basta con un simple copiado y su diagrama es el que sigue:



6.3.9.- Mezclar bases de datos

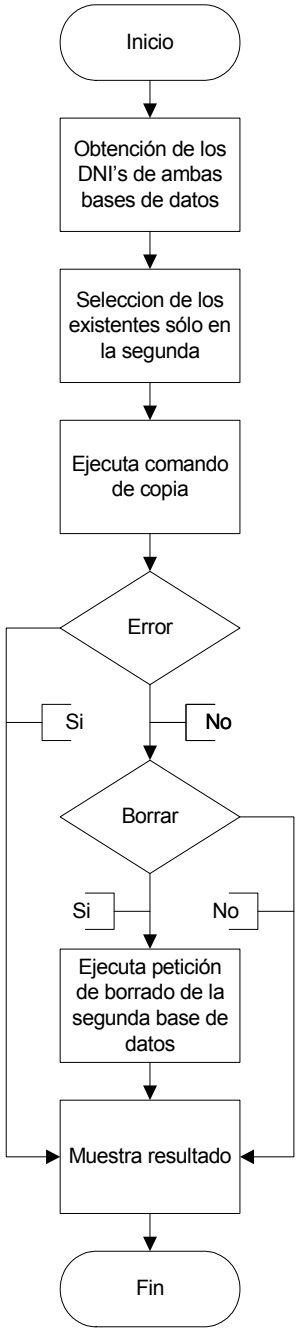
Esta opción permite intercambiar o copiar expedientes entre dos bases de datos, para ello se muestra un listado de los expedientes contenidos en cada una, permitiendo el movimiento de los mismos de una a otra, luego se efectúa la actualización de ambas. De la implementación de la función se hace cargo el VI TScr Mix_DB



Como se observar el proceso de intercambio se basa en la copia de expedientes de entre las bases de datos, seguida de un borrado posterior en caso de haber solicitado la opción mover.

6.3.10.- Unir bases de datos

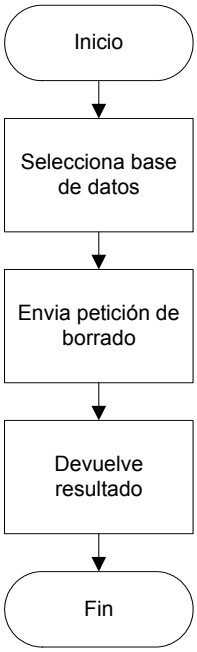
Esta opción permite unir dos bases de datos en una, mediante el copiado de los datos de la segunda a la primera, de tal modo que en caso de datos repetidos prevalecerán los de la primera.



Una vez finalizado el proceso se preguntara si se desea eliminar la base de datos que se unió.

6.3.11.- Eliminar base de datos

Esta opción permite eliminar una base de datos. Su funcionamiento es muy simple, ya que solo pide el nombre de la base de datos y la elimina.



Mencionar que estas operaciones se solicitan por parte del terminal pero se ejecutan en el servidor, de ahí la necesidad de las peticiones en lugar de un procesado directo.

También destacar que en ciertas opciones se realiza un precargado de la última búsqueda de expedientes realizada con el gestor de éstos, a fin de facilitar la selección de expedientes a mover, copiar, dividir, etc.

6.3.12.- Seleccionar base de datos

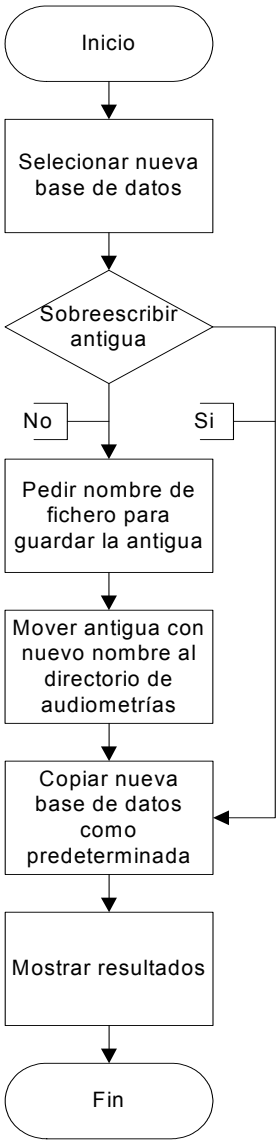
Después de generar todos los archivos de gestión de bases de datos, surge la necesidad de una opción que permita escoger la base de datos con la que el programa trabaja.

En principio el usuario podría elegir sobre que base de datos desea realizar sus operaciones, pero con el kit de LabView empleado, hay que realizar una configuración de la base de datos en los controladores de orígenes de datos (ODBC) del sistema operativo en el que trabaja. Esta configuración ha de ser realizada de manera manual y directa en el sistema operativo en el que opere el programa, por tanto no es posible ir creando bases de datos y luego seleccionarlas de manera directa, pues no estarían cargadas en el ODBC del sistema.

La solución es simple, el sistema necesita de dos bases de datos, una de trabajo, y otra para junto a la primera permitir la realización de las operaciones de copia, mezcla, división,... de bases de datos. Por tanto cargaremos en el ODBC ambas bases de datos.

Ahora, si solo se utilizan estas dos, el problema esta en como hacer que se pueda utilizar una base de datos distinta a una de estas. Bien, para solucionar esto se plantea el siguiente modo de trabajo. Se configuran las dos bases de datos en el ODBC como las bases de datos de trabajo del SGA, y cuando se desee seleccionar otra base de datos para trabajar con ella, se sobrescribirá esta sobre la que se emplea actualmente, de manera que los datos que aparecen son los de la nueva base de datos, pero el sistema operativo no es capaz de percibir el cambio, ya que mantiene el nombre, tipo y demás parámetros.

La rutina que se obtiene según estas pautas figura en la página siguiente.


Inicio
Selecionar nueva base de datos
Sobreescribir antigua
No
Si
Pedir nombre de fichero para guardar la antigua
Mover antigua con nuevo nombre al directorio de audiometrías
Copiar nueva base de datos como predeterminada
Mostrar resultados
Fin

6.3.13.- Expedientes Médicos

Esta aplicación se encarga de la gestión de los expedientes médicos, y es por tanto la parte más importante de la interfaz, ya que el resto solo se encargan de la gestión del sistema cuya finalidad es permitir a los usuarios interaccionar con estos expedientes médicos. Luego esta aplicación será la que más tiempo de ejecución acapare en un proceso normal de interfaz.

La aplicación que soporta esta característica es independiente de la interfaz de usuario y se aloja en el VI Scr DB_Control

Las variables que se almacenan dentro de un expediente son múltiples y muy variadas, lo que denota un sistema bastante complejo. Además el hecho de que el LabView fuese diseñado para aplicaciones de control y no para aplicaciones de interfaz con el usuario complica aún más el método de programación, recuérdese que se han de utilizar bucles infinitos de chequeo, mientras se espera a que el usuario introduzca los datos.

Para gestionar los expedientes se ha optado por un control denominado Tab de datos, en él, los datos se ordenan por carpetas, dentro de las cuales pueden meterse los controles que se estime oportunos.

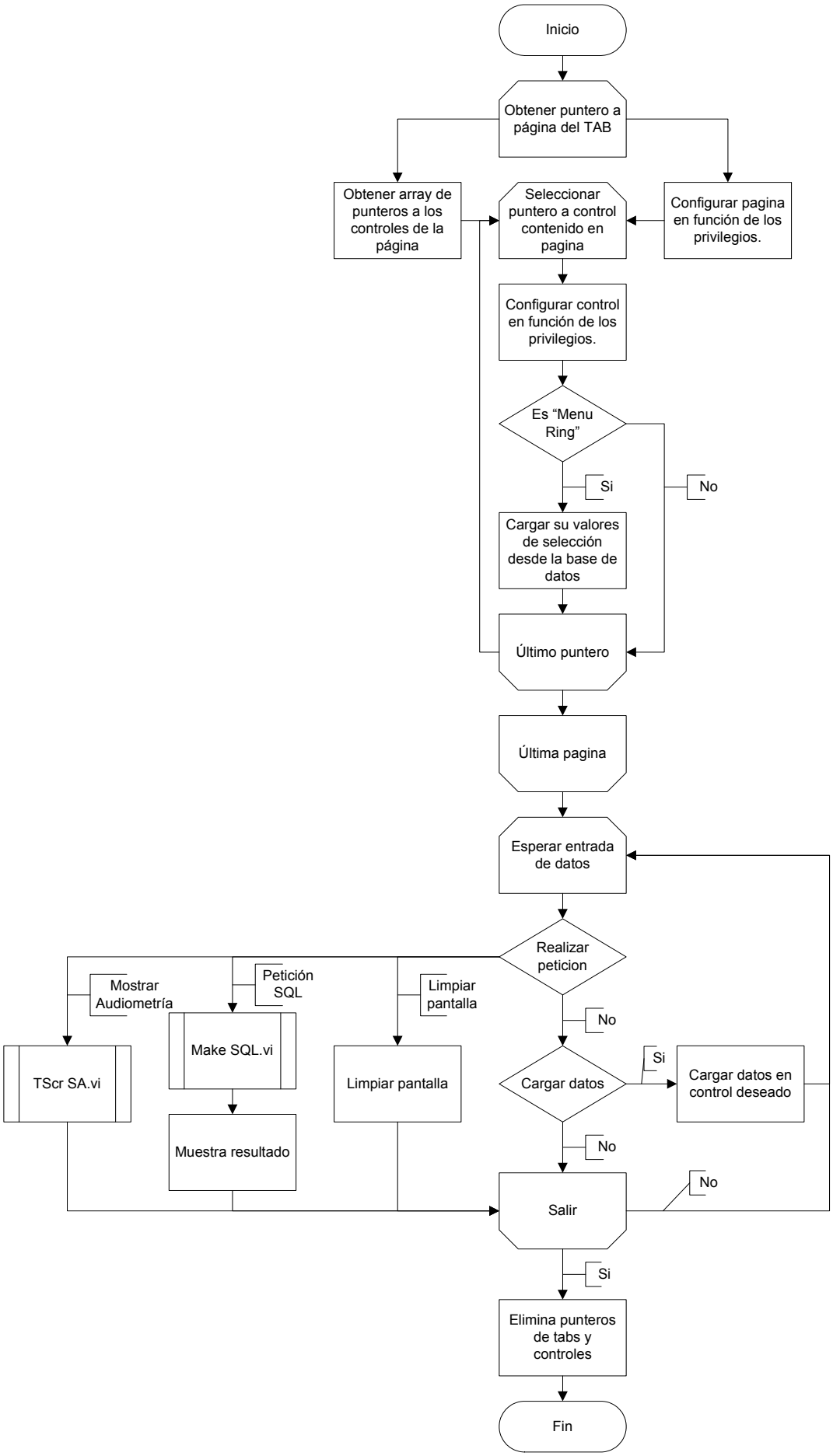
Este tipo de control permite asignar una carpeta a cada tabla de datos, y luego añadir dentro de la carpeta, tantos controles como datos contenga dicha tabla.

El hecho de que los usuarios tengan distintos tipos de privilegio a la hora de acceder a estos expedientes, complica aun más el sistema, al tener que mantener un estricto control de accesos.

El dato principal por el que se identifican los usuarios, es por el DNI numérico, es decir se omiten las letras, si el usuario utilizase otro tipo de código de identificación (NIE, Pasaporte, etc...), se introduciría del mismo modo pero podría dar lugar a solapamientos, por lo que es preferible mantener un único tipo, que seria el DNI o en su defecto NIE.

Al iniciarse la aplicación, lo primero que se realiza es un chequeo de permisos, restringiendo ciertas carpetas para que el usuario, en función a sus privilegios, no pueda seleccionarlas o modificarlas. Esto no significa que las carpetas no estén ahí, sino todo lo contrario, el sistema no conoce a que carpetas o controles tiene acceso el usuario y trabajara del mismo modo para unos permisos que para otros, sin embargo, el usuario no podrá abrir una carpeta para la que no tenga permiso de lectura, y podrá abrirla pero no manipular sus controles si sólo tiene permiso de lectura pero no de escritura. Obviamente, el permiso de escritura implica obligatoriamente el de lectura.

Este manejo de privilegios impide la creación de expedientes, si se carece de permiso de escritura para datos personales, ya que el identificador principal es el DNI, y sin permiso de escritura en datos personales no puede introducirse dicho dato principal. Sin embargo, si el usuario fue creado anteriormente, si es posible modificar sus datos sin necesidad de tener permiso de escritura en datos personales.


Inicio
Obtener puntero a página del TAB
Obtener array de punteros a los controles de la página
Seleccionar puntero a control contenido en pagina
Configurar pagina en función de los privilegios.
Configurar control en función de los privilegios.
Es "Menu Ring"
Si
No
Cargar su valores de selección desde la base de datos
Último puntero
Última pagina
Esperar entrada de datos
Realizar peticion
Mostrar Audiometría
Petición SQL
Limpiar pantalla
No
TScr SA.vi
Make SQL.vi
Limpiar pantalla
Cargar datos
Si
Cargar datos en control deseado
Muestra resultado
No
Salir
No
Si
Elimina punteros de tabs y controles
Fin

En la página anterior aparece el esquema correspondiente a esta aplicación. Como se observa el programa a pesar de ser complicado en codificación, presenta un esquema más o menos simple.

En el instante inicial se genera una lista (array) bidimensional de punteros a las distintas carpetas (dimensión principal), y a los controles existentes dentro de cada carpeta (dimensión secundaria), la creación del array se sistematiza con un bucle sobre un puntero principal que apunta al TAB y del cual se pueden obtener los punteros a cada uno de estas carpetas y controles. Dentro de este primer bucle se genera otro que analiza uno a uno los controles contenidos en la página y que se describe en el siguiente párrafo, pero que no es necesario para la obtención del array bidimensional ya que con el primer bucle se obtiene de cada página un array a cada uno de sus controles en el orden en que fueron creados al realizar el programa. Esto en principio no es muy importante pues los comandos SQL no tienen porque seguir un orden estricto, sino que basta con que a la hora de crear el comando, las columnas y los datos concuerden en orden (la primera columna en el comando se ha de corresponder con el primer dato en el comando, y así con las siguientes columnas y datos, pero la primera columna no tiene porque corresponderse a la primera columna de la tabla sobre la que se realiza el comando, basta con que exista dentro de ella). Sin embargo, hay ciertos casos en los que el orden es importante, por ejemplo, el caso de datos personales, en el que aparece un objeto OLE (la foto), difícil de manejar dentro de la tabla de datos personales y que por tanto se ha de tratar de manera especial, siendo interesante que aparezca en último lugar. LabView permite modificar el orden de estos controles dentro de un panel, por lo que no hay excesivo problema, pero simplemente recordar esta particularidad y ajustar dicho orden.

Como se comentaba en el párrafo anterior, mientras se genera el array bidimensional, se realiza un segundo bucle mediante el cual se cargan los valores de selección de esos controles especiales denominados “Menu Rings”, esto es una función añadida a nuestro sistema por la que el numero de elementos que contiene uno de estos controles es modificable, y para ello basta con añadir más campos o menos en una tabla dentro de la base de datos, que esta dedicada a los campos de estos elementos de control. Así, es posible tener un control con un listado de enfermedades, e incrementar o decrementar dicho listado de enfermedades sin tener que modificar la codificación del programa. Bastaría con añadir o eliminar el campo dentro de la tabla de datos correspondiente. Esto tiene la clara ventaja de dar control al cliente sobre los datos que desea que aparezcan, de una manera muy sencilla (la tabla base de datos puede modificarse con MS Access, aunque podría añadirse un elemento de configuración dentro del sistema). Por último sobre esta característica, destacar que el hecho de eliminar un campo no actualiza los datos de todos los expedientes por lo que si un paciente se creó con un campo determinado de enfermedad, y luego se elimina ese campo del control correspondiente en el TAB, los expedientes antiguos que contuvieran dicho campo, lo mantienen, mientras que los nuevos carecerán de la posibilidad de introducir dicho campo como enfermedad.

El uso de los dos bucles anteriores es también necesario para la configuración de los controles y páginas en función de los privilegios. Como resultado de esa configuración es posible obtener un control, habilitado y visible, deshabilitado y visible, o deshabilitado y no visible.

En la página anterior se ha descrito la iniciación realizada por este programa, sin embargo, aún no se sabe nada de su funcionamiento normal.

Una vez iniciado, el programa comienza a interactuar con el usuario, esto como ya se ha mencionado varias veces, implica un bucle continuo de espera de entrada de datos, en este bucle no se realiza nada mientras el usuario no pulse un botón de control, una vez realizada dicha acción, el sistema comprueba que tipo de acción ha solicitado.

Si solicitó un limpiado de pantalla, se borran todos los valores contenidos en los controles del TAB.

Si por el contrario solicitó la carga de datos dentro de un control tipo tabla, se realiza dicha carga.

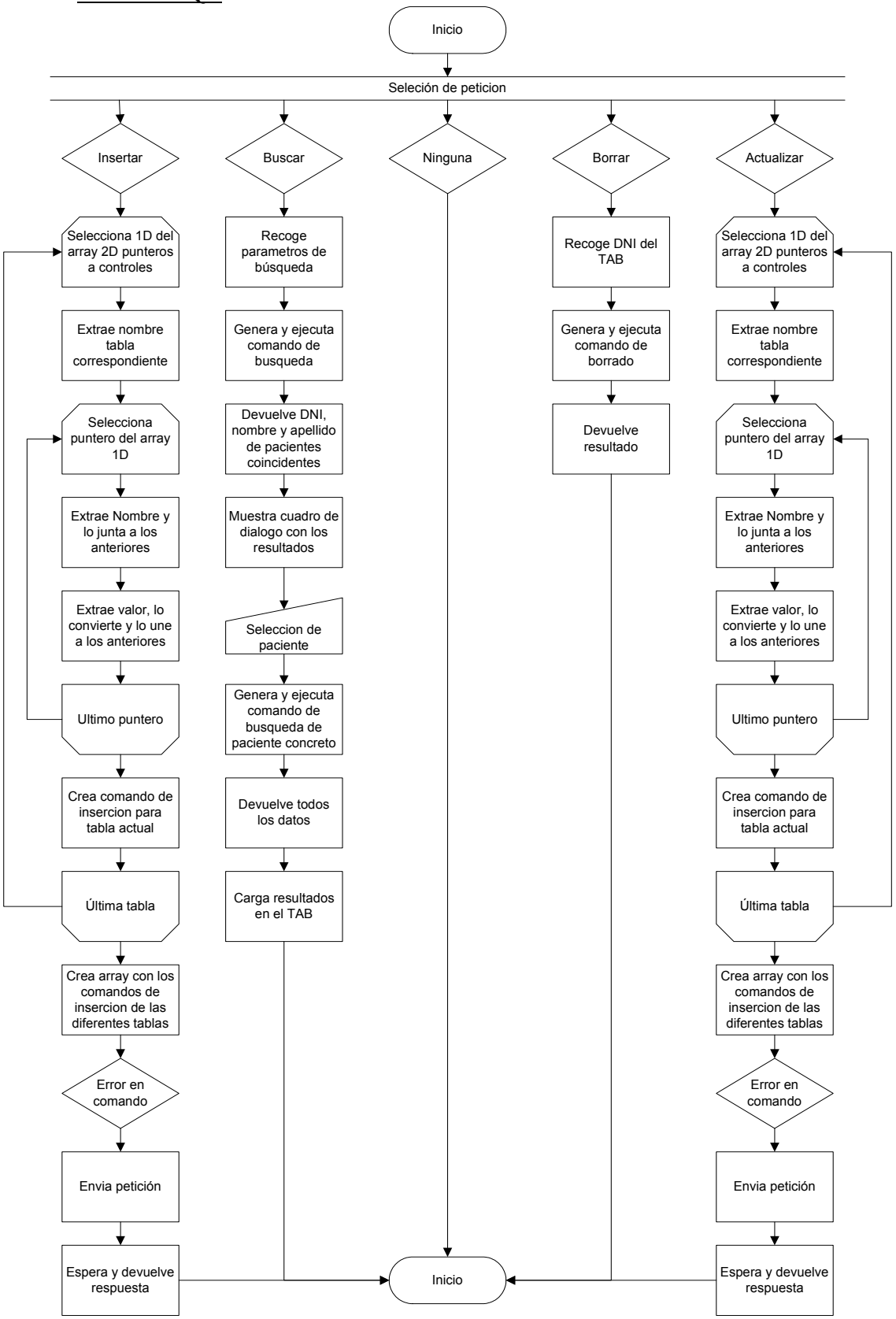
Si solicitó la realización de una operación sobre la base de datos, se procede a la implementación y ejecución del comando SQL correspondiente. Este proceso se realiza en una subrutina almacenada dentro del VI Make SQL, ya que el sistema en si, posee muchos controles y es bastante complicado como para no delegar dichas operaciones. Aun así, hay que distinguir este ultimo VI de otros anteriores como TScr C_User, TScr M_User, ... ó incluso el propio TScr DB_Control, ya que éstos se lanzan de manera independiente al VI User Interface, mientras que Make SQL se lanza de manera interna y como subrutina de Make SQL, eso implica un interacción directa, así como una sencilla transmisión de datos, lo que no descarta la necesidad de introducir ciertos punteros dentro de las variables globales para hacer disponible a la subrutina, cierta información contenida en elementos de la rutina principal.

Si solicitó visualizar una audiometría lanza el VI TScr SA, que se encarga de cargar los datos correspondientes a la audiometría seleccionada y visualizarlos.

Por último si lo que solicito fue cerrar el panel de control de expedientes, se procede al cierre del VI.

Explicar la concordancia entre nombres porque sino no se va a entender nada del siguiente apartado.

VI Make SQL:


Inicio
Seleción de peticion
Insertar
Buscar
Ninguna
Borrar
Actualizar
Selecciona 1D del array 2D punteros a controles
Extrae nombre tabla correspondiente
Selecciona puntero del array 1D
Extrae Nombre y lo junta a los anteriores
Extrae valor, lo convierte y lo une a los anteriores
Ultimo puntero
Crea comando de insercion para tabla actual
Última tabla
Crea array con los comandos de insercion de las diferentes tablas
Error en comando
Envia petición
Espera y devuelve respuesta
Recoge parametros de búsqueda
Genera y ejecuta comando de busqueda
Devuelve DNI, nombre y apellido de pacientes coincidentes
Muestra cuadro de dialogo con los resultados
Seleccion de paciente
Genera y ejecuta comando de busqueda de paciente concreto
Devuelve todos los datos
Carga resultados en el TAB
Recoge DNI del TAB
Genera y ejecuta comando de borrado
Devuelve resultado
Selecciona 1D del array 2D punteros a controles
Extrae nombre tabla correspondiente
Selecciona puntero del array 1D
Extrae Nombre y lo junta a los anteriores
Extrae valor, lo convierte y lo une a los anteriores
Ultimo puntero
Crea comando de insercion para tabla actual
Última tabla
Crea array con los comandos de insercion de las diferentes tablas
Error en comando
Envia petición
Espera y devuelve respuesta
Inicio

En la página anterior está el diagrama de la aplicación que genera los comandos SQL, que se envían al servidor a fin de que en conjunto realicen la petición solicitada.

La primera cosa a mencionar es que en el apartado de búsqueda la manera en que se generan los comandos de búsqueda y obtención de datos, es idéntica a la de inserción, o actualización, lo único que varía es el comando (insert, search, update), pero la forma en que se generan los parámetros de búsqueda es igual en los tres, y en que en la búsqueda se devuelven resultados distintos a un código de error. La razón por la que no se ha incluido todo el proceso dentro de la figura es la falta de espacio.

Una vez hecha la aclaración pertinente, se procederá a la explicación del funcionamiento de la opción de inserción. En ella se observa como primero se separa el array bidimensional dejando una única dimensión que corresponde a los punteros de una misma carpeta, el nombre de la carpeta ha de ser el mismo que el de la tabla a la que corresponde dentro de la base de datos (mencionar que como en otro caso anterior, no es necesario que el orden de las carpetas sea el mismo que el de las tablas, pero es imprescindible que los nombres coincidan) y está almacenado en una variable global. Con el nombre de la tabla se pasa a procesar el array obteniendo de cada control su nombre, que ha de corresponderse con el de la columna, y su valor, los nombres se juntan por un lado y los valores por el otro, una vez terminado el proceso se genera el comando que al ser de tipo SQL suele seguir el siguiente patrón:

Comando (tabla) Modificador (Columnas) Modificador (Datos)...

Este patrón puede no cuadrar exactamente con ninguno de los comandos SQL, pero en los rasgos más generales sí, y es justo lo que quiero resaltar, ya que sirve para dejar constancia de porque se agrupan por un lado los nombres y por otro los datos. Si se desea información más concreta sobre los comandos SQL puede consultarse el apartado dedicado a tal fin.

Volviendo al comando generado, como puede verse corresponde a la primera tabla, pero el primer bucle volverá a interactuar seleccionando el segundo array del array bidimensional, para procesarlo y así con cada uno de ellos hasta obtener un comando por cada tabla.

En un principio, el sistema solo insertaba datos en aquellas tablas cuyas carpetas correspondientes contuvieran algún dato, sin embargo esto generaba un serio problema. Ciertas carpetas del mismo expediente contenían datos, mientras que otras carecían de ellos, de modo que la siguiente vez que se intentase trabajar con el expediente, se debería utilizar el comando insertar en las tablas vacías, y el comando modificar en las tablas rellenas.

Este problema daba serios quebraderos de cabeza, pues los chequeos cargaban el servidor y complicaban las rutinas de generación de comandos SQL, las cuales estaban ya bastante sobrecargadas por la conversión de los datos del programa a datos compatibles con los contenidos en la base de datos.

La solución era obvia, el comando tenia que ser ejecutado en todas las tablas y no únicamente en aquellas que contenían datos. Como el código identificativo principal de un expediente era el DNI, se decidió que al insertar, todas las tablas contendrían como base el DNI del paciente, y luego el resto de campos que tuvieran rellenos, de este modo siempre podría actualizarse sin el temor a encontrar una tabla vacía.

Esta solución llevó a la idea de la necesidad de que el servidor permitiese que le enviasen estos comandos juntos, ya que sino, alguien que estuviese utilizando el mismo expediente, podría mezclar sus comandos con los nuestros produciendo errores indeseados. Añadiendo esa opción en el servidor, el resultado es que todos los comandos solicitados se ejecutan juntos, es decir secuencialmente pero uno tras otro, como un conjunto, de modo que si alguien desea realizar otro comando será también como conjunto y antes o después del nuestro, pero nunca a la vez. Esto puede dar un error a alguien que intenta actualizar un expediente, que justo antes ha sido borrado, pero este tipo de errores, son poco habituales y carecen de importancia al mantener la estructura interna de los datos intacta.

El comando actualizar se implementa de manera muy similar al de insertar, lo único que cambia es el comando SQL así como los modificadores, que son típicos de cada comando, pero su forma de obtención seguiría la misma pauta. Se crearían los comandos individuales para cada tabla, separando por un lado el nombre de sus controles y por otro el valor almacenado en los mismos, y luego se juntarían todos los comandos a fin de que al enviarse al servidor, éste a partir del conjunto de comandos simples realizase la actualización total del expediente.

La acción buscar difiere ya del procedimiento aplicado a las anteriores, aunque la generación de comandos es similar.

Para buscar, lo primero que se hace es generar un comando de búsqueda siguiendo la misma pauta que en los dos comandos anteriores. Este comando de búsqueda toma los datos del expediente y busca expedientes que al menos contengan esos datos, pudiendo el resto ser distintos. Además, el comando se configura para que devuelva los siguientes datos de DNI, nombre y apellidos, de aquellos expedientes que concuerden con el buscado.

Una vez realizada la búsqueda se muestran los datos que corresponden a todos aquellos expedientes que coinciden con el patrón buscado. En el panel de visualización el usuario tiene la opción de seleccionar uno, el cual vuelve a ser buscado para obtener todos sus datos y cargarlos en el TAB, sin embargo la búsqueda esta vez es mucho más sencilla, pues basta con buscar los datos de cada tabla correspondientes al DNI seleccionado y devolverlos para proceder a su carga.

Tras realizar el comando se cargan los datos en el TAB y quedan listos para consulta o actualización.

En principio el sistema permite realizar cualquier opción, pero se recomienda que para buscar, e incluso para borrar, se realice primero una búsqueda del expediente, puesto que es posible que al actualizar se escriba sobre datos, que al no verlos puedan parecer poco importantes cuando en realidad son cruciales. A la hora de insertar no hay problema, ya que si el expediente está registrado, el sistema dará un error de inserción.

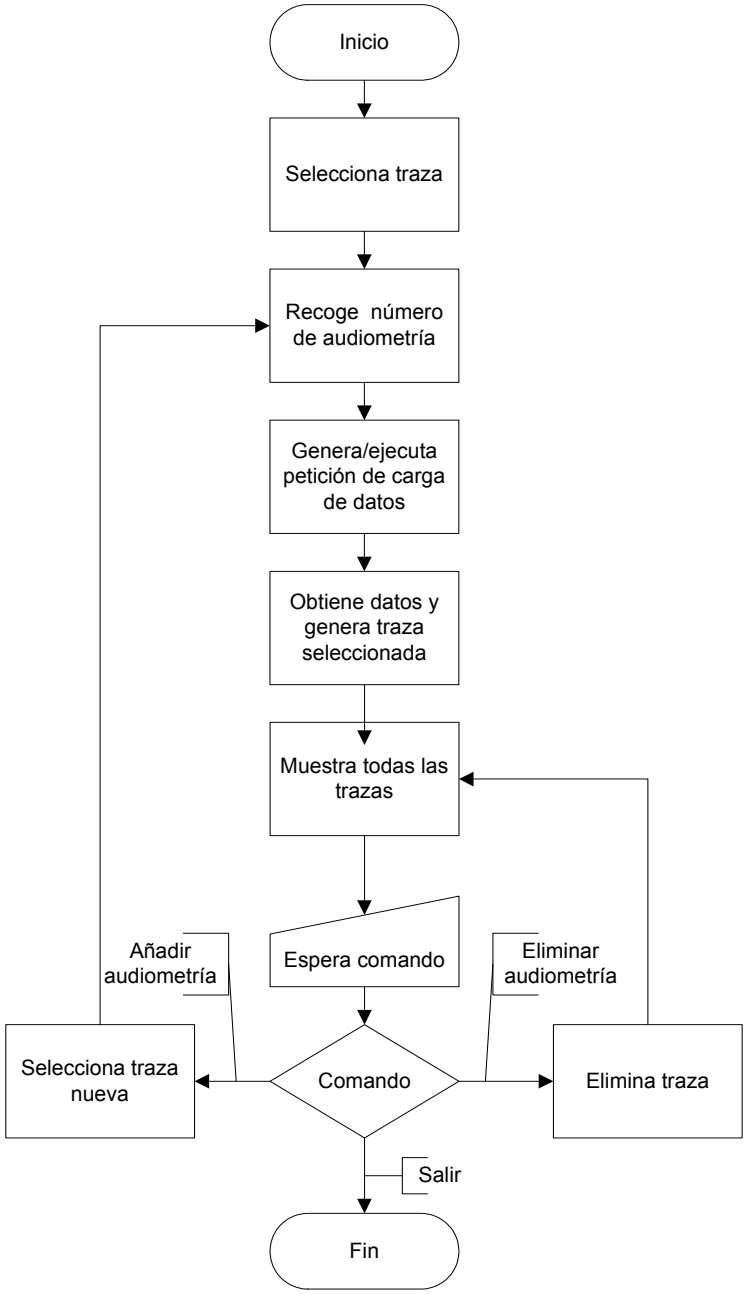
El comando borrar es el más simple de todos, puesto que solo se permite borrar por el parámetro DNI, si no se supiese el DNI del expediente a ser borrado, basta con hacer una búsqueda previa, con lo que se cargará el DNI en el TAB y luego proceder al borrado.

El comando de borrado por DNI se genera de la misma manera que el de búsqueda por DNI, ahora en vez de dos bucles (uno para tabla y otro para controles) basta con realizar uno para las tablas e introducir en el comando que el único valor a tener en cuenta es el del DNI y que ha de coincidir con aquel registrado en el TAB.

Este programa recibe del VI TScr DB_Control la opción que se desea realizar, pero por si en algún momento alguien lo reutilizase, o por algún fallo anormal de este VI se le enviase otra opción distinta de las cuatro reseñadas, el programa finalizaría su ejecución sin dar ningún tipo de error, ignorando simplemente el comando que no reconoce.

VI TScr SA:

El VI TScr SA es el encargado de visualizar audiometrías. Dentro de nuestro TAB con el expediente del paciente, existe un listado de las audiometrías que le han sido realizadas, junto con algunos datos concretos sobre dichas audiometrías. Sin embargo, los valores registrados en el examen audiométrico no se cargan al expediente desde la base de datos, sino que quedan pendientes de que se solicite la visualización de la audiometría. De este modo se ahorra un tiempo de proceso que en principio no es totalmente útil, pues es posible que el usuario no necesite visualizar todas las audiometrías, e incluso que no necesite visualizar ninguna, por lo que cargando todas aumentaría el tiempo de proceso y la sobrecarga del servidor sin ninguna garantía de su necesidad. Es por eso que se ha optado por cargar los datos cuando se solicite la visualización de una audiometría, esto ralentiza el proceso de visualizado, pero teniendo en cuenta que solo será necesario cargar los datos de una única audiometría la perdida de velocidad es imperceptible.

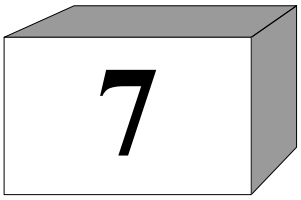


A la hora de implementar este VI se disponía de dos opciones, la primera era lanzarlo como subrutina de TScr DB_Control, con lo que la transferencia de datos sería directa pero el control de expediente quedaría bloqueado a la espera de la finalización del visualizador de audiometrías para continuar con su ejecución normal. Esto en principio no era un gran problema, pues la verdad es que no es muy normal dejar audiometrías abiertas mientras se consultan otros expedientes, pero aún así cabía otra opción que era posiblemente más interesante. Esta segunda opción proponía el lanzado independiente de la aplicación por lo que se necesitaría una comunicación a través de variables globales, pero permitiría trabajar simultanea e independientemente con ambos programas. Son estas características de mayor dinamismo y libertad de ejecución las que decantaron la balanza a favor de segunda opción.

El programa visualiza en primer lugar la audiometría seleccionada en el gráfico, sin embargo los datos de las audiometrías no se identifican por el DNI y número de audiometría, sino que lo hacen únicamente con el número de audiometría hecho que hace que el visualizador desconozca el paciente al que pertenecen dichas audiometrías no pudiendo ofrecer la carga sobre el mismo grafico del resto de audiometrías pertenecientes al mismo expediente.

Para evitar esto, se envía, a través de una variable global, el listado de audiometrías pertenecientes al paciente, de este modo el visualizador ofrece tanto la adición al gráfico del resto de audiometrías pertenecientes al paciente, como la adición de audiometrías por el número de audiometría, esto último da la posibilidad de comparar en el mismo gráfico audiometrías de distintos pacientes.

La audiometría inicial a visualizar, es recibida también vía variable global, siendo ésta última un array que permite enviar más de un número de audiometría para agilizar la visualización de varias audiometrías al no tener que ir añadiéndolas una a una. Esta opción se tiene también dentro de la adición al gráfico de más audiometrías del propio paciente, si bien no es posible emplearla para la visualización de audiometrías de otro paciente.



LA BASE DE DATOS

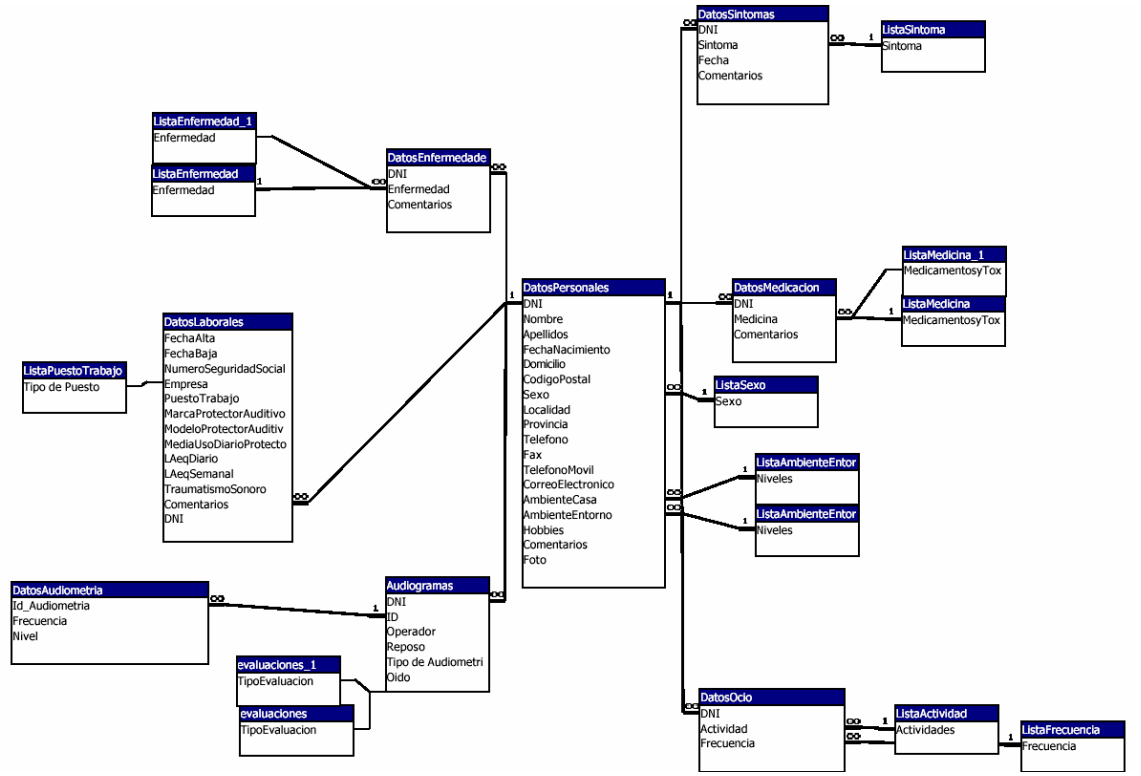
- Estructura.

- Características.

- Peculiaridades.

7. LA BASE DE DATOS:

En la siguiente figura se muestra el modelo relacional de la base de datos. En los anexos se ha incluido una figura de mayor tamaño.



Como se observa, todas las tablas principales dependen del campo DNI excepto la de datos de audiometría que se vincula únicamente con la tabla de audiogramas al corresponderse un identificador de audiogramas concreto.

También existe una serie de tablas que se asocian a campos concretos y cuyo nombre se compone de "Lista...", todas estas tablas están vinculadas a un control del tipo "Menu Ring" en el SGA y simplemente contienen los datos que ese control debe presentar dentro del SGA. Para ayudar a la inserción manual de datos en Access y para mejorar la visualización de las relaciones, se han vinculado también dentro de la base de datos, a aquellos campos que se tratan con controles del tipo "Menu Ring".

Como ya se mencionó la base de datos hay que meterla dentro del ODBC del sistema operativo, y los comandos se ejecutan en todas las tablas, mediante una petición especial que permite ejecutarlas como conjunto, ya que en principio, funciones como inserción, actualización y borrado, sólo pueden ejecutarse sobre una tabla (la función de búsqueda sí es posible ejecutarlas sobre múltiples tablas).

En principio se trabajó con la base de datos como algo estático, pero poco a poco se fue haciendo coincidir ciertos valores entre la base de datos y el TAB que permitían dinamizarlo bastante.

El hecho de utilizar punteros para determinar la estructura de la tabla, a partir de la estructura del TAB, permitía que una modificación en la tabla sólo necesitase de una modificación en el TAB, para que el programa continuase funcionando sin detectar el cambio, y sin generar ningún tipo de error. Esto era debido a que los bucles detectaban el puntero nuevo y efectuaban una iteración más.

Para ello era necesario hacer coincidir el nombre de las carpetas y de las tablas de tal modo que las funciones que generaban los comandos para tablas anteriores, pasaban a hacer lo propio con la tabla nueva.

Claro está que esas funciones no dependían únicamente de los nombres de las tablas y carpetas, sino también de los controles contenidos en la carpeta y los campos contenidos en la tabla. El problema era idéntico al de las tablas, al hacer aparecer el mismo número de controles en la carpeta que el número de campos en la tabla y además coincidentes en nombre, el resultado era intachable, ya que los punteros se ajustaban a los campos y los bucles al número de iteraciones necesarias.

Con este nuevo sistema el dinamismo era total, añadir, borrar, modificar campos en las tablas o incluso añadir, borrar, modificar tablas era cosa sencilla, no eran necesarios cambios internos en la programación bastaba con añadir una nueva carpeta con sus controles y el programa quedaba funcionando.

Las funciones que generaban los comandos distinguían automáticamente los tipos de datos y los ajustaban a aquellos contenidos dentro de la base de datos, de modo que tampoco este ámbito daba ningún problema.

Sin embargo había ciertas tablas, en las que un campo de un paciente podía tener más de un valor almacenado. Estos nuevos tipos de campo se trataron en el TAB como controles tipo “tables”, y se generaron las rutinas pertinentes para su funcionamiento, salvaguardando el dinamismo, una vez realizadas las concordancias entre campos múltiples en la base de datos, con “tables” de datos en el TAB.

Un último caso problemático era el de los controles OLE, en este proyecto había uno, el campo foto de paciente. Este campo se trató en principio como un campo externo a la base de datos. Las fotos se guardaban en disco duro, y se buscaban en un directorio de fotos, a partir del DNI.

Sin embargo, se generó una rutina muy específica que permitía introducir dichas fotos dentro de la base de datos como objetos OLE, sin comprometer el dinamismo, pero introduciendo un caso particular adicional al de los controles tipo “table”.

Entonces se intentó dar una vuelta más a la programación y conseguir una tabla totalmente dinámica, el procedimiento consistía en emplear unos VI’s que permitían introducir cualquier tipo de dato en cualquier campo de modo que podía crearse un TAB con el formato de carpetas y controles deseados. Para esto el programa lo primero que hacía era comprobar si el formato de nuestro TAB y el de la base de datos coincidía, si no generaba una base de datos acorde al TAB, a sus carpetas y a sus controles. Una vez hecho esto el programa manejaba todo tipo de datos sin tener ningun problema especial con los controles tipo “table” o controles tipo “OLE”, todos los comandos de ejecución, inserción y borrado eran independientes del tipo de datos, y por tanto ya no era

necesario crear carpetas coordinadas con las tablas de la base de datos, sino que al crear un TAB cualquiera para el programa, el sistema generaba su propia base de datos, siendo innecesario ningún conocimiento de programación para ampliar o modificar el sistema. Ya que sólo era necesario modificar el aspecto de un panel de control LabView.

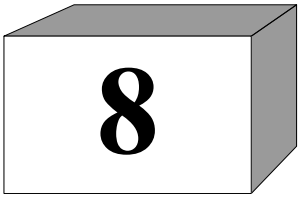
El sistema obtenido era completamente dinámico, automático y ajustable de una manera extremadamente simple, sin embargo, no era posible realizar búsquedas.

El programa era capaz de buscar, expedientes idénticos, e incluso expedientes en los que los campos simples fuesen similares, pero era incapaz de encontrar campos "table" por patrón, o se introducía todos los datos del control exactamente como se encontraban en la base de datos, o el simple hecho de cambiar el orden de aparición conseguía confundir al buscador y no reconocer que el expediente coincidía con el patrón buscado.

Este problema de búsqueda no pudo solucionarse de ninguna de las maneras, por lo que el sistema actual es el anterior, un poco menos dinámico, con necesidad de crear tablas y campos coordinados con carpetas y controles (y no sólo carpetas y controles) para ampliar o modificar funciones, y con funciones específicas para objetos OLE y "table", pero capaz de realizar las búsquedas según patrón, sin ningún tipo de problema, incluso en estos casos particulares.

Por tanto, las ampliaciones o modificaciones en los expedientes, precisan de ampliaciones o modificaciones en la base de datos, pero ya no es necesario modificar la programación interna del SGA. Es decir el programador, no necesitará conocer a fondo el sistema sobre el que realiza la modificación pero si necesitará conocer la herramienta de gestión de bases de datos Access.

De todos modos, aunque el sistema fuese capaz de modificar las bases de datos en función de los cambios de estructura en su TAB, sería necesario adaptar los expedientes antiguos a esta nueva estructura, por lo que la última solución, aunque hacia una programación plana, rápida y sin casos particulares, no llegaba a librar de la necesidad de conocimientos de Access.



8 CONCLUSIONES, PRUEBAS Y MEJORAS

- Pruebas
- Mejoras
- Conclusiones

8.1. MEJORAS

El desarrollo de este proyecto, implicaba la generación de una aplicación nueva, sin basarse en ninguna anterior. Para ello, se fueron desarrollando rutinas especificas para la realización de ciertos procesos y que daban soporte a una interfaz simple.

Según el proyecto avanzaba las funciones se complicaban cada vez más mientras que la interfaz se consideraba demasiado simple.

A partir de ese momento se comenzó a revisar el código de los distintos procedimientos a fin de hacerlos, más simples, más intuitivos, procurando que siguieran las mismas pautas, convirtiendo su procesado en algo similar e independiente del procedimiento concreto. Sustituyendo particularidades y casos concretos, por procedimientos comunes, logrando un gran grado de automatización de procesos.

A esa nueva codificación se agregó una paulatina mejora de la interfaz con el usuario, procurando que ésta se volviese algo más inteligente, ahorrando tiempo al usuario, y a la vez facilitando la introducción de datos y el manejo de los expedientes.

Se introdujeron rutinas de detección y solución de errores, rutinas de configuración del sistema, etc.

8.2 PRUEBAS

Se ha realizado pruebas del sistema, tanto en la parte del servidor, como en la parte del terminal, del mismo modo que se superviso el funcionamiento conjunto de ambas.

Durante todas esas pruebas se fue comprobando el correcto funcionamiento de cada elemento integrante de la aplicación, siendo corregidos todos los errores encontrados.

La gran cantidad de entradas y posibilidades hacen del funcionamiento algo difícil de comprobar, sin embargo, el hecho de que las opciones del sistema actuasen sobre los mismos elementos, permitía detectar una gran serie de fallos, que si bien, en un principio no eran fáciles de detectar mediante la supervisión de la aplicación que los generaba, el uso del resto de aplicaciones orientadas a los mismos servicios terminaba por manifestarlos y una vez observados se localizaban fácilmente, acotándose y procediéndose a su solución.

Las pruebas a realizar sobre una aplicación tan sumamente amplia, son muy abundantes, sin embargo, cada una de las aplicaciones desarrolladas así como las subrutinas que las integran han sido testadas de manera paulatina, antes de proceder al desarrollo de la siguiente. Y como se ha mencionado el propio uso y desarrollo de las nuevas funciones que se iban implementando permitía aumentar la cobertura de errores.

8.3 CONCLUSIONES

Se ha cumplido con las especificaciones impuestas al principio del desarrollo de este proyecto, las aplicaciones han sido verificadas tanto a nivel local como de red de más de un terminal.

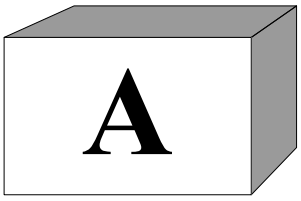
A las especificaciones iniciales se han ido agregando otras impuestas por los deseos de los desarrolladores y del tutor, a fin de obtener una herramienta de mayor potencia y prestaciones llegándose a un resultado bastante satisfactorio.

La aplicación ha sido desarrollada en distintos sistemas operativos, adaptándose de manera sencilla entre unos y otros por lo que indica un buen grado de compatibilidad. Del mismo modo LabView permite la recompilación de las aplicaciones para compatibilizarlas de manera absoluta con otros sistemas operativos, lo que garantiza el funcionamiento de esta aplicación en cualquier sistema.

Este nuevo sistema de gestión de audiometrías esta realizado con LabView, lo que hace de su integración en el sistema global que permite la realización automática de audiometrías, algo sencillo y transparente.

Su sencillez, hace de la documentación algo innecesario, al ser su uso algo muy intuitivo para cualquier tipo de usuario.

LabView es una herramienta de programación que esta en constante desarrollo, y por lo que ha presentado ciertos errores internos a la hora de ejecutar el programa (problemas con las DNS que obligaron al uso de direcciones IP, problemas con los datos datasocket que impidieron la utilización de arrays tridimensionales, etc.) que han obligado muchas veces a la modificación de la implementación del sistema, sin embargo dichos errores inherentes a LabView, que se van eliminando con la aparición de nuevas versiones, no lograron impedir la consecución de los objetivos.



ANEXOS

- SQL

- Base de datos

ANEXO I: SQL

1.- INTRODUCCION

El lenguaje de consulta estructurado *(SQL)* es un lenguaje de base de datos normalizado, utilizado por el motor de base de datos de Microsoft Jet. *SQL* se utiliza para crear objetos QueryDef, como el argumento de origen del método *OpenRecordSet* y como la propiedad *RecordSource* del control de datos. También se puede utilizar con el método *Execute* para crear y manipular directamente las bases de datos Jet y crear consultas *SQL* de paso a través para manipular bases de datos remotas cliente - servidor.

1.1. Componentes del SQL

El lenguaje *SQL* está compuesto por comandos, cláusulas, operadores y funciones de agregado. Estos elementos se combinan en las instrucciones para crear, actualizar y manipular las bases de datos.

1.2 Comandos

Existen dos tipos de comandos *SQL*:

- Los *DLL* que permiten crear y definir nuevas bases de datos, campos e índices.
- Los *DML* que permiten generar consultas para ordenar, filtrar y extraer datos de la base de datos.

Comandos DLL

CREATE: Utilizado para crear nuevas tablas, campos e índices.

DROP: Empleado para eliminar tablas e índices.

ALTER: Utilizado para modificar las tablas agregando campos o cambiando la definición de los campos.

Comandos DML

SELECT: Utilizado para consultar registros de la base de datos que satisfagan un criterio determinado.

INSERT: Utilizado para cargar lotes de datos en la base de datos en una única operación.

UPDATE: Utilizado para modificar los valores de los campos y registros especificados.

DELETE: Utilizado para eliminar registros de una tabla de una base de datos.

1.3 Cláusulas

Las cláusulas son condiciones de modificación utilizadas para definir los datos que desea seleccionar o manipular.

Cláusulas

FROM: Utilizada para especificar la tabla de la cual se van a seleccionar los registros.

WHERE: Utilizada para especificar las condiciones que deben reunir los registros que se van a seleccionar.

GROUP BY: Utilizada para separar los registros seleccionados en grupos específicos.

HAVING: Utilizada para expresar la condición que debe satisfacer cada grupo.

ORDER BY: Utilizada para ordenar los registros seleccionados de acuerdo con un orden específico.

1.4 Operadores Lógicos

AND: Es el "y" lógico. Evalúa dos condiciones y devuelve un valor de verdad sólo si ambas son ciertas.

OR: Es el "o" lógico. Evalúa dos condiciones y devuelve un valor de verdad si alguna de las dos es cierta.

NOT: Negación lógica, devuelve el valor contrario de la expresión.

1.5 Operadores de Comparación

<	Menor que
>	Mayor que
<>	Distinto de
<=	Menor ó Igual que
>=	Mayor ó Igual que
=	Igual que
BETWEEN	Utilizado para especificar un intervalo de valores
LIKE	Utilizado en la comparación de un modelo
IN	Utilizado para especificar registros de una base de datos

2.- CONSULTAS DE SELECCION

Las consultas de selección se utilizan para indicar al motor de datos que devuelva información de las bases de datos, esta información es devuelta en forma de conjunto de registros que se pueden almacenar en un objeto recordset. Este conjunto de registros es modificable.

2.1 Consultas básicas

La sintaxis básica de una consulta de selección es la siguiente:

SELECT Campos FROM Tabla;

En donde campos es la lista de campos que se deseen recuperar y tabla es el origen de los mismos, por ejemplo:

SELECT Nombre, Teléfono FROM Clientes;

Esta consulta devuelve un recordset con el campo nombre y teléfono de la tabla clientes.

2.2 Ordenar los registros

Adicionalmente se puede especificar el orden en que se desean recuperar los registros de las tablas mediante la cláusula ORDER BY Lista de Campos. En donde Lista de campos representa los campos a ordenar. Ejemplo:

SELECT CodigoPostal, Nombre, Telefono FROM Clientes ORDER BY Nombre;

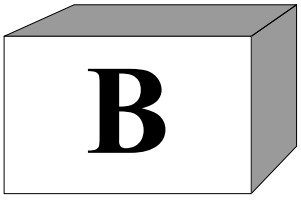
Esta consulta devuelve los campos CodigoPostal, Nombre, Telefono de la tabla Clientes ordenados por el campo Nombre.

Se pueden ordenar los registros por mas de un campo, como por ejemplo:

SELECT CodigoPostal, Nombre, Telefono FROM Clientes ORDER BY CodigoPostal, Nombre;

Incluso se puede especificar el orden de los registros: ascendente mediante la cláusula (ASC) ó descendente (DESC)

SELECT CodigoPostal, Nombre, Telefono FROM Clientes ORDER BY CodigoPostal DESC , Nombre ASC;



BIBLIOGRAFÍA

- Bibliografía

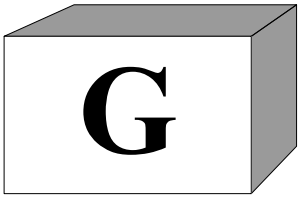
- Páginas Web

BIBLIOGRAFÍA:

- LabView Basics I Course Manual.
- Database connectivity Toolset User Manual.
- El estándar ANSI SQL (86). Pedro Alarcón Cavero (Dic–94) [OEI–EUI–UPM]

PÁGINAS WEB:

- www.ni.com



Glosario

- Definiciones breves.

BREVE GLOSARIO

- **Array:** Conjunto de N variables del mismo tipo que se crean con un mismo nombre, distinguiéndose por la posición que ocupan dentro del conjunto. Un array es una variable estática, es decir, se crea con un determinado numero de elementos, y aunque se utilicen menos, todos los elementos están ocupando memoria.

- **Base de datos:** Una base de datos es una colección de datos operacionales utilizados por todas las aplicaciones de una organización, una serie de tablas que contienen información ordenada en alguna estructura que facilita el acceso a esas tablas, ordenarlas y seleccionar filas y de las tablas según criterios específicos. Su utilidad: permite utilizar un único sistema centralizado de almacenamiento de la información.

- **Cola:** Estructura de memoria, en la que se pueden colocar datos para posteriormente ser utilizado. En este proyecto se utilizan para la comunicación interna de aplicaciones.

- **Control:** Elemento de un panel de LabView, con el que el usuario puede interactuar, y que se emplea como variables de entrada de datos para las aplicaciones LabView.

- **Control tipo "table":** Tipo específico de control, en el que los datos se organizan en filas y columnas.

- **Datasocket:** Protocolo de transmisión que se monta sobre TCP/IP, estableciendo unas colas de datos, en las que se envían o reciben datos entre los usuarios de las colas.

- **Login:** Acción de entrada en un sistema que por seguridad requiere un registro de acceso.

- **Logout:** Acción de abandono de un sistema en el que previamente se hizo un login.

- **Menu Rings:** Tipo concreto de control, que permite al usuario la selección, entre un determinado número de valores concretos.

- **Objeto Ole:** Objeto en formato estándar que permite que sea utilizado por otras aplicaciones distintas a las específicas de dicho objeto.

- **Panel:** Elemento de un VI LabView, mediante el cual se presentan al usuario los controles sobre los que puede actuar.

- **SQL:** Véase anexo I.

- **TAB:** Tipo específico de LabView, que permite clasificar los controles en carpetas, presentando al usuario distintos conjuntos de controles en función de la carpeta seleccionada.

- **TCP/IP:** Protocolo común utilizado por todos los computadores conectados a Internet, de manera que éstos puedan comunicarse entre sí. Es compatible con cualquier sistema operativo y hadware.

- **VI:** Instrumento Virtual LabView, es un conjunto de Panel y programa de gestión del panel. En programación debe entenderse como un programa o subrutina.

ANEXO II: BASE DE DATOS